



Bilkent University

Department of Computer Engineering

Senior Design Project

Final Report

Project Name: BookClub

Group Members: Bikem Çamlı
Mert Osman Dönmezyürek
Barış Eymür
Mahin Khankishizade
Deniz Şen

Supervisor: Assoc. Prof. Selim Aksoy

Jury Members: Prof. Dr. Fazlı Can
Assoc. Prof. Cigdem Gunduz Demir

Innovation Expert: Dr. Haluk Altunel

Final Report

May 9, 2019

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491/2

Table of Contents

| | |
|---|-----------|
| Acronyms and Abbreviations | 5 |
| 1. Introduction | 5 |
| 1.1 Engineering and Report Writing Standards | 6 |
| 1.2. Use of New Tools and Technologies | 6 |
| 1.3. Lifelong Learning | 7 |
| 1.4. Implementing Creative and Impactful Solutions | 7 |
| 2. System Overview | 8 |
| 3. BookClub's Algorithmic and Application Design | 10 |
| 3.1. Server Algorithms and Implementation | 10 |
| 3.1.1. Match Algorithm | 10 |
| 3.1.2. Suggestion Algorithm | 11 |
| 3.2. User Experience and Interface Design | 13 |
| 4. Final System Architecture Design | 14 |
| 4.1. Subsystem Overview | 14 |
| 4.2. Packages | 17 |
| 4.2.1. Client Package | 17 |
| 4.2.1.1 View | 18 |
| 4.2.1.1.1 Layout | 18 |
| 4.2.1.2 Controller | 19 |
| 4.2.1.3 Model | 19 |
| 4.2.2 Server Package | 20 |
| 4.2.2.1 Model | 21 |
| 4.2.2.2 Controller | 21 |
| 4.2.2.2.1 ServiceManager | 21 |
| 4.2.2.2.2 AlgorithmManager | 21 |
| 4.2.2.2.3 ModelManager | 22 |
| 4.3 Class Interfaces | 22 |
| 4.3.1 Client | 22 |
| 4.3.1.1 View | 24 |
| 4.3.1.1.1 Layout Package | 24 |
| 4.3.1.1.1.1 WelcomeActivity Class | 25 |
| 4.3.1.1.1.2 LoginActivity Class | 25 |
| 4.3.1.1.1.3 ForgotPasswordActivity Class | 26 |
| 4.3.1.1.1.4 TradeBookActivity Class | 26 |
| 4.3.1.1.1.5 SignUpActivity Class | 26 |

| | |
|---|-----------|
| 4.3.1.1.1.6 MainActivity Class | 27 |
| 4.3.1.1.1.7 ChatListActivity Class | 28 |
| 4.3.1.1.1.8 ChatActivity Class | 28 |
| 4.3.1.1.1.9 MatchListFragment Class | 29 |
| 4.3.1.1.1.10 SuggestionListFragment Class | 29 |
| 4.3.1.1.1.11 PreferencesActivity Class | 29 |
| 4.3.1.1.1.12 HistoryActivity Class | 30 |
| 4.3.1.1.1.13 RequestBookActivity Class | 31 |
| 4.3.1.1.1.14 TradeListActivity Class | 31 |
| 4.3.1.1.1.15 AccountSettingsActivity Class | 32 |
| 4.3.1.1.1.16 WishlistActivity Class | 33 |
| 4.3.1.1.1.17 GeneralListFragment Class | 33 |
| 4.3.1.1.1.18 BookInfoActivity Class | 34 |
| 4.3.1.2 Controller | 34 |
| 4.3.1.2.1 BookClubAPI Class | 34 |
| 4.3.1.3 Model | 38 |
| 4.3.1.3.1 Book Class | 38 |
| 4.3.1.3.2 User Class | 39 |
| 4.3.2 Server | 40 |
| 4.3.2.1 Controller | 42 |
| 4.3.2.1.1 AlgorithmManager Package | 42 |
| 4.3.2.1.1.1 AlgorithmController Class | 42 |
| 4.3.2.1.2 ServiceManager Package | 42 |
| 4.3.2.1.2.1 EmailServiceController Class | 42 |
| 4.3.2.1.2.2 ChatServiceController Class | 43 |
| 4.3.2.1.3 ModelManager Package | 43 |
| 4.3.2.1.3.1 UserController Class | 43 |
| 4.3.2.1.3.2 WishListController Class | 44 |
| 4.3.2.1.3.3 HistoryController Class | 45 |
| 4.3.2.1.3.4 TradeListController Class | 45 |
| 4.3.2.1.3.5 AccountSettingsController Class | 46 |
| 4.3.2.1.3.6 ChatController Class | 47 |
| 4.3.2.1.3.7 MessageController Class | 47 |
| 4.3.2.2 Model | 47 |
| 4.3.2.2.1 User Class | 47 |
| 4.3.2.2.2 WishList Class | 48 |
| 4.3.2.2.3 History Class | 49 |

| | |
|---|-----------|
| 4.3.2.2.4 TradeList Class | 49 |
| 4.3.2.2.5 AccountSettings Class | 50 |
| 4.3.2.2.6 Chat Class | 50 |
| 4.3.2.2.7 Message Class | 51 |
| 4.3.2.2.8 Book Class | 51 |
| 4.3.2.2.9 Match Class | 52 |
| 4.3.2.2.10 Suggestion Class | 52 |
| 5. Impact of Engineering Solutions | 53 |
| 5.1. Economical Impacts | 53 |
| 5.2. Environmental Impacts | 53 |
| 5.3. Social Impacts | 54 |
| 6. Tools and Technologies Used | 54 |
| 6.1. Development Tools | 54 |
| 6.1.1 Pycharm | 54 |
| 6.1.2. Visual Studio Code | 54 |
| 6.1.3. GitHub | 55 |
| 6.1.4. Postman | 55 |
| 6.1.5. Xampp | 55 |
| 6.1.6. phpMyAdmin | 55 |
| 6.1.7. HeidiSQL | 56 |
| 6.1.8. Trello | 56 |
| 6.1.9. Android Studio | 56 |
| 6.1.10. Genymotion Emulator | 57 |
| 6.2. Libraries and Frameworks Resources | 57 |
| 6.2.1 Square Picasso | 57 |
| 6.2.2 Swipe to Dismiss Undo | 57 |
| 6.2.3 Spots Dialog | 57 |
| 6.2.4 LovelyDialog | 58 |
| 6.2.5 Pandas | 58 |
| 6.2.6 Factory-boy and Faker | 58 |
| 6.2.7 Django | 58 |
| 6.2.8 Django-cors-headers | 58 |
| 6.2.9 djangorestframework | 59 |
| 6.2.10 mysqlclient | 59 |
| 6.2.11 Android SDK 28 | 59 |
| 7. Engineering Solutions and Contemporary Issues | 59 |

| | |
|--|-----------|
| 7.1 Internet Related Issues | 59 |
| 7.2 Issues Related to Huge Market Size | 60 |
| 7.3 Language and Communication Issues | 60 |
| 7.4 Data Privacy Issues | 60 |
| 7.5 Security Issues | 61 |
| 8.User Manual | 62 |
| 8.1 Initial Main Page | 62 |
| 8.2 Login Page for Normal Users | 63 |
| 8.3 Forgot Password Page | 64 |
| 8.4 Login or Register Proposal Page for Guests | 65 |
| 8.5 Register Page | 66 |
| 8.6 General Page For Users | 67 |
| 8.7 General Page for Guests | 68 |
| 8.8 Matches Page | 69 |
| 8.9 Suggestions Page | 70 |
| 8.10 Settings Page | 71 |
| 8.11 Account Settings Page | 72 |
| 8.12 Wishlist Page | 73 |
| 8.13 Tradelist Page | 74 |
| 8.14 History Page | 75 |
| 8.15 Add New Book to Wishlist Page | 76 |
| 8.16 Add New Book to Tradelist Page | 77 |
| 8.17 My Chats Page | 78 |
| 8.18 Single Chat's PagePage | 79 |
| 8.19 Rating a User Page | 80 |
| 8.20 Other User's Profile Page | 81 |
| 8.21 Own Profile Page | 82 |
| References | 83 |

Acronyms and Abbreviations

UML: Unified Modeling Language

API: Application Programming Interface

MVC: Model View Controller design pattern

JSON: JavaScript Object Notation

ORM: Object-relational mapping

1. Introduction

Reading is a fun and useful activity: we can learn new things by reading a scientific book or we can find ourselves in a totally new fictional world by reading a novel. Unfortunately, every beautiful thing comes with a price. Wouldn't it be nice to exchange our books with our friends' books, instead of paying? If we did that, we would have lessened the price of reading significantly. There are groups of people around the world who have noticed the advantages of book sharing. These people have formed book sharing groups, called book clubs.

The main aim of this project is to create a book trading platform which will serve as a universal "BookClub". The users of this mobile app will be able to exchange their used books with books of other users. The users will specify the books that they desire to give away and the ones they want to read. Using this information, BookClub will match users with the owners of books they are looking for. While doing this, it will find the best possible match for a user by considering user locations and user ratings. Additionally, it will make personal suggestions to its users.

This report starts with BookClub's system overview. Then, the implementation details of BookClub's match and suggestion algorithms and the user interface design are given. Following this, details of our system architecture are explained, using package diagrams and class interfaces. The report continues with economical, environmental and social impacts of BookClub. After this section, the development tools, frameworks and libraries

we used in implementation are given and the way we used them in the development process are explained. Then, contemporary issues related with the BookClub's project domain is included. The report ends with the user manual for BookClub.

1.1 Engineering and Report Writing Standards

All the diagrams and figures that describe software or hardware compositions included in the reports of BookClub follow the UML guidelines. We chose to use UML because all of our group members were proficient in it, since it was taught in CS 319 (Object Oriented Software Engineering) course.

The citations and references in all reports follows IEEE's standards. These standards are highly recommended for use in the engineering reports, thus we decided to use them.

1.2. Use of New Tools and Technologies

RestAPI: We implemented the API between client and server with the help of RestAPI. All the requests were handled via this API. [1]

Django Python Framework: We used this framework in the server-side because it is very easily adaptable to MVC pattern and easily controls the model side with its controllers. Additionally, the overall structure of the Django Framework was well suited for our project, since BookClub is a database driven mobile app.[2]

JSON Data: As our data type, we used JSON to communicate between client and server. This kind of data comes very handy and works efficiently with RestAPI.

Django Python ORM: Since we used Django Python Framework, instead of directly writing MySQL queries, we decided to use Django's own ORM. This ORM came very handy in creating queries because of its syntax. With this ORM the complex queries were very easy to implement.

Postman: We used Postman to send requests to server functions and test them while we were developing our API. We simulated the client side in Postman by sending necessary requests to server functions and checking if the necessary operations are performed successfully. [3]

AndroidAPI 28 Nougat: As BookClub is an Android application, we used Android SDK to implement it and we used the newest one which is API 28.

Android Studio: Android Studio is the IDE that Google themselves recommend for the development of Android applications. It is based on IntelliJ Idea which is full of useful functions that aid the developer during the coding process.[4]

NetBeans: NetBeans IDE was used during the development of the client-server API. It also contains many useful features for Java development.

OkHTTP3: OkHTTP3 is a modern library that makes http request handling easier and more efficient. [5]

1.3. Lifelong Learning

The BookClub users will have an opportunity of getting original books easily and for free. By using BookClub, they will be able to explore new books with BookClub's innovative suggestion algorithm and getting their desired books with the best match algorithm. The best part of these transactions are - they are completely free. By this way, users of BookClub will always be able to find books to read, which will give them the chance to constantly reach new information.

The developers of BookClub, will constantly update the application by using the user feedback. Additionally, they will try to adjust the application to the newest technologies and environments.

1.4. Implementing Creative and Impactful Solutions

In implementation of BookClub we have mainly considered two things: the communication between client and server team and the user experience. Both of these values had to be at their best.

In order to improve the communication between two teams, we had some standards in the implementation of the code. First of them was to write understandable and readable code with the comments as headers. Additionally, when something was changed in the

API or server part, the client team was being notified via third party apps such as Trello, Google Drive or WhatsApp group chats.

In order to improve the user experience within the app, we constantly were testing the app with the people who has no idea what it is about. Their feedback was very useful in enhancing the user experience and changing the interface of the application.

Additionally, we tried to find an external server, because the application was going to be an Android application, achievable from all Android devices.

2. System Overview

Let's imagine a case where a user wants to read/buy a book, however, the book is either too expensive, not available in the local markets or worse - the book is only obtainable as a pirate version. In this situation, let's think about what are the primary websites/applications the user will use to obtain the original version of the book for a more reasonable price.

First, they will search "buy [book name] cheap" in Google. Later, they will click the link leading to Amazon and observe that the book they want is second-hand, original and cheap. However, it is unknown whether the book will be shipped safely in a good condition. In this case, the user may not be satisfied and will search other websites/applications.

The user may take a look at online second-hand book markets such as ThriftBooks, BookFinder, etc [6], [7]. Nevertheless, although in these kinds of websites, the books are cheap, the shipping price is usually high and the sellers/owners are unreachable. Additionally, the location of the book is unspecified. At this point, the user may either buy the pirate version or not be able to obtain the original book at all. This is the problem which leads to black marketing, devaluation of the books and authors.

BookClub is an innovative mobile application which can be downloaded for free from Google Play Store. The main difference of BookClub from the similar applications/websites is that in this app, users can exchange the book they desire with a unique match/suggestion algorithm. The users log in/register to the app and create the

lists of the books they can give away and the books they desire to obtain. Then, the user will get matched with other users. In this situation, by clicking the match screen, they will see the list of matches and the owners. They can easily contact the user and arrange a meeting where they will be able to exchange books. This is an advantage for both sides since the users will be worried about neither the safety of the book nor its condition, due to being able to reach them by chat. There is no shipment price and no worries about the reliability of the other. Additionally, the possibility that the user will get the pirate version of the book is really low.

There are also the cases of not getting the best match, in this case, BookClub's innovative suggestion algorithm offers users new books that they may like, by analyzing their book wishlist and tradelist. The suggestion algorithm tries to find the books to suggest similar to the user's wishlist and tradelist. For example, if the user is interested in the books of genres detective, there is a high chance that the algorithm will suggest something similar to this genre. Additionally, the suggestion algorithm analyzes the user's tradelist. For instance, if the user gives away fiction genre books, it means that the user likes this genre if they have bought it once. Thus, the suggestion algorithm will suggest some books similar to the fiction.

Both match and suggestion algorithm get affected and lead by some factors such as the distance between user's, the trader's rating, the book's position in the wishlist, etc. In other words, the user's each action changes the algorithm and makes it better.

Overall, BookClub is an innovative and accessible mobile app that will create an opportunity for book lovers to find new books or trade their books with a simpler method. The aim of the app is to ease book lovers' lives by creating a convenience for them to swap their original books or get them for a cheaper price. With the original and fast match/suggestion algorithm it will decrease both the black market sales and book piracy. At the same time, the app will provide the users the books they desire.

3. BookClub's Algorithmic and Application Design

3.1. Server Algorithms and Implementation

3.1.1. Match Algorithm

Giving our users best possible matches for them is one of the main functionalities of BookClub. In order to achieve this, we implemented our match algorithm by considering locations and ratings of users.

We decided to search matches for a user in a circular area with radius of 50 kms, by placing the user that we are searching for matches at the center of this circle. We store the locations of our users in the “bookclub_server_user” table of our database as longitude (long) and latitude (lat) values. In our distance related calculations, we used the information that the distance between 2 latitudes are 111 kms everywhere, whereas the distance between 2 longitudes vary from latitude to latitude (since all longitudes merge at the polar points). Since we developed the initial release of our app for use in Turkey, and since Turkey resides between the northern latitudes of 36° and 42°, we decided to use the distance between 2 longitudes in our match algorithm as the distance between 2 longitudes on the northern latitude of 40°, which is 86 kms. Using these distance metrics, our algorithm obtains the users that are at most 50 kms distant from our user and we search for matches with these users. The reason for us to implement this limitation was, we believe that if a user has a match with someone who is more than 50 kms away from him/her, then this match would not be practical, since trading the books will be hard in this scenario.

Then, in this pool of users, we try to find matches by first checking if these users have any books to give that our user wants to obtain. We perform this by comparing the books in our user's wishlist with the books in other users' tradelists. In BookClub, a match is a two-way process, which means that when User A desires a book that User B can give, then User A must also have a book to give that User B wants to obtain. Therefore, we

also compare the books in our user's tradelist to books in other users' wishlists. After this, we obtain our matches.

The next task that our algorithm performs is to calculate scores for these matches. We calculate scores in a scale of 0 to 100 (as integer). 50% of our match score is based on the location point of the matched user and 50% of the match score is based on the user rating point of the matched user. Our algorithm finds the location point by subtracting the distance between the 2 users from 50. It determines the user rating point by calculating the average of rating values for the matched user (these rating values are out of 5) and multiplying it with 10. By default, our algorithm takes the rating of a user who is not rated yet as 2.5. Then, we add the location point and the user rating point to each other and obtain the match score.

3.1.2. Suggestion Algorithm

The aim of our suggestion algorithm is to generate the best suggestions for a user by using the books in his/her wishlist. By this way, users get the chance to receive possible trades that are suggested by BookClub to them by considering many factors.

Similar with our match algorithm, our suggestion algorithm obtains the users that are in a circular area with radius of 50 kms from our user. Then, it calculates the ratings for these users and find the top 3 users with highest ratings among them. We decided to create recommendations only from the 3 users with the highest ratings, because otherwise we observed that the number of recommendations generated was quite large, which was unnecessary.

Then, by comparing the books in our user's wishlist with the books in other users' tradelists, we obtain our first suggestion scores. There are 4 criteria that our algorithm uses to calculate suggestion points; which are user locations, user ratings, attribute similarity between books and common books that exist in different users' wishlists. Our suggestion scores are in the range of 0 to 100. The suggestion score is calculated in the following way:

| Point | Percentage in Suggestion Score |
|----------------------------------|--------------------------------|
| Book Attributes Similarity Point | 30% |
| User Rating Point | 25% |
| User Location Point | 25% |
| Wishlist Similarity Point | 20% |

In determining the similarity of attributes between books, we compare the authors and publishers of 2 books. These were the only attributes in our dataset which can be used to compare the similarity between 2 books. Book attributes similarity point is initially calculated out of 50. If the author of 2 books are the same, then the similarity point is 30; if the publisher of the books are the same, the similarity point is 20. If both the authors and the publishers of the 2 books are identical, then the similarity point is 50, which is the maximum value. We convert the book attributes similarity point to a value out of 30 after these calculations.

Wishlist similarity point reflects the tendency of other users who want the same book in their wishlist with our user to also want the book which is suggested by BookClub. If many other users who want Book A also wants Book B, then it may point out that these 2 books can be related with each other. Wishlist similarity point is calculated out of 10 in the following way:

- If number of wishlists that contain both Book A and Book B is greater than or equal to 4, then the wishlist similarity point is 10.
- If number of wishlists that contain both Book A and Book B is equal to 3, then the wishlist similarity point is 7.
- If number of wishlists that contain both Book A and Book B is equal to 2, then the wishlist similarity point is 4.
- If number of wishlists that contain both Book A and Book B is equal to 1, then the wishlist similarity point is 1.

- If number of wishlists that contain both Book A and Book B is equal to 0, then the wishlist similarity point is 0.

We convert the wishlist similarity point to a value out of 20 after these calculations.

The user rating point and the user location point are calculated in the same way as in our match algorithm. We convert both the user rating point and the user location point to values out of 25. Then, we add these 4 types of points up and obtain our first suggestion score. If this score is greater than or equal to 50, then this suggestion is saved for future consideration. Otherwise, we do not take this possible trade as a suggestion, since its score is too low. We perform this operation between every book in our user's wishlist and every book in other 3 users' tradelists.

After this process is over and all of the first suggestion scores are calculated, our algorithm then performs all of these operations again, in order to determine the second suggestion scores. The second suggestion scores compare the books that are in our user's tradelist and the books which are other users' wishlists. The reason for this is that, all trades in BookClub are 2-way processes, therefore when a user receives a book, he/she should also give a book in return. Then, we eliminate the possible trades which has second suggestion score below 50, and save the possible trades which have second suggestion score greater than or equal to 50 for future consideration.

Then, by comparing the first and second suggestion scores and finding the possible trades with common users, the algorithm generates suggestions. The final suggestion scores are calculated by taking the average of the first and second suggestion scores for a particular trade. We decided to hold the maximum number of generated suggestions as 5 for a single execution of the algorithm, because if we did not give a limit for the number of suggestions, then the algorithm was creating too many suggestions, which is an unnecessary and impractical situation.

3.2. User Experience and Interface Design

The user interface was designed according to several rules that were defined by our team. For instance, every card design in the application have exactly the same corner

radia and same elevation values. These are achieved by defining style values which are used in several different usage points. On the other hand, the user interface design of BookClub does not include complex algorithms, mostly, the algorithms are run on the server side of the application, sent via the API, and the resulting data are adapted to the screen using custom adapters and different asynchronous tasks that are run on the background. In this context, it is necessary to point out the fact that the synchronization of the UI thread and the worker thread is mainly done inside AsyncTask class. The task separates the UI thread and the worker thread, but it is the coder that synchronizes these kinds of threads.

4. Final System Architecture Design

4.1. Subsystem Overview

Following page contains the package diagram of BookClub.

The system architecture of BookClub can be described as a Client / Server architecture. We chose to implement our platform using this architecture because we want BookClub to be able to deal with a large number of users without any deteriorations in performance. Thanks to our server, BookClub will be able to serve many users (clients) in a fast and successful way. We use the server for many purposes. We store data about books, users and user book lists in the server. This will give us a significant opportunity to keep data safe. Also, the user match algorithm will be executed in the server side to generate matches and suggestions for users. Moreover, the server side will be responsible for performing most of the computations. The Android app, which will be the client side, will perform basic operations. It will connect to the server, request relevant information from the server and display it to the user. Then, the users will be able to make their own choices, such as accepting or rejecting a match, etc. The data in the server side will be updated according to the choice being made on the client side. With this architectural structure, BookClub will be able to work in a fast and efficient way and continue to perform its function without any deteriorations, even if the number of users increases significantly.

In BookClub, we also use Model-View-Controller (MVC) design pattern. We distributed parts of MVC between client and server sides. Client-side contains View, which contains classes about user interface and user interaction.

Server-side contains Model (user and book data that is stored in a database) and Controller (which acts as a bridge between Model and View by coordinating and manipulating the user input in the application).

The users of BookClub will interact with the View part of the Client and this will be the only part of the application that the users will be able to directly interact with. The users' input will be taken through the view part. They will be sent to the server including the data and the user's request via ConnectionManager class in the Client.

4.2. Packages

4.2.1. Client Package

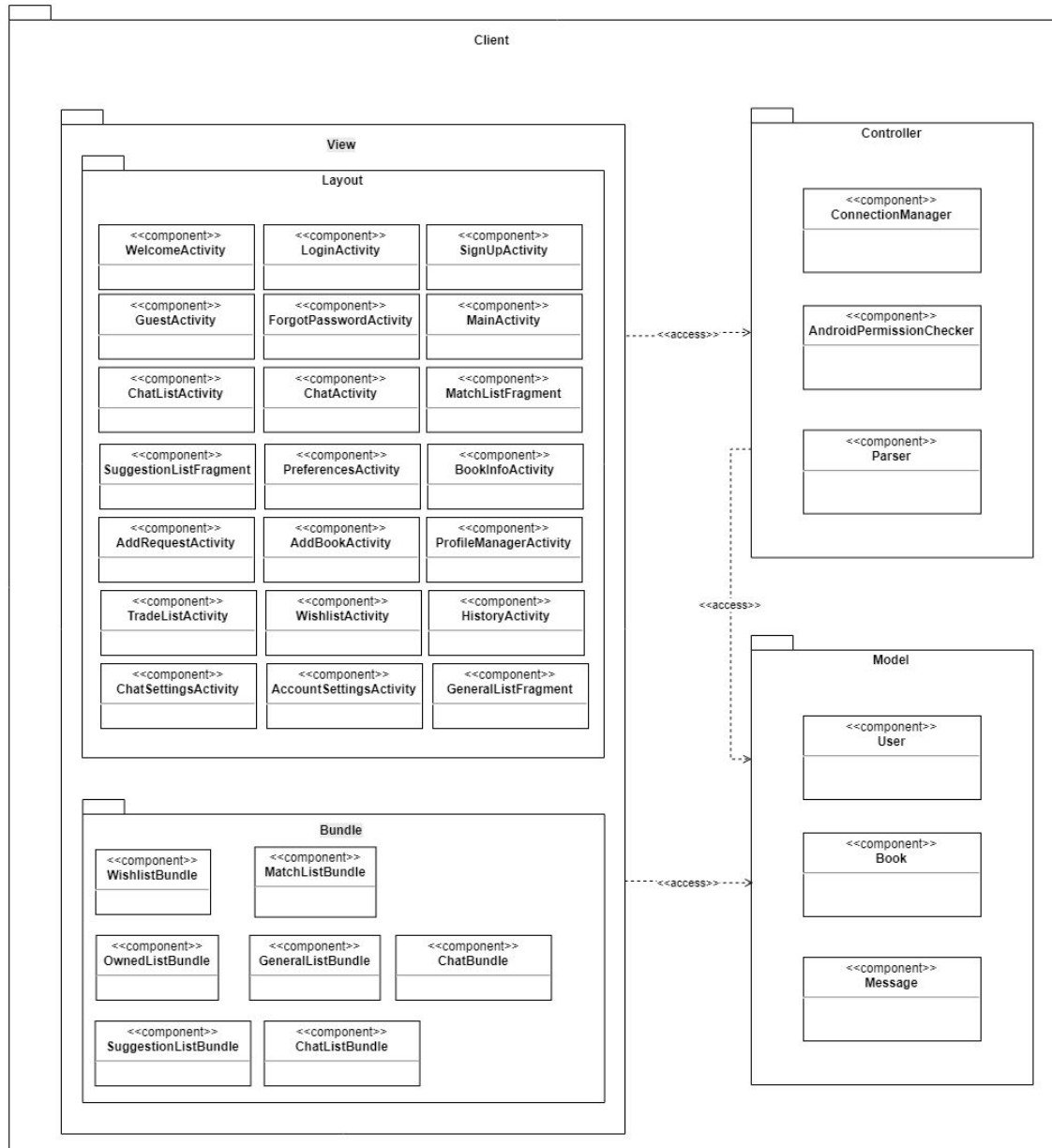


Figure: Package Diagram of Client-Side

4.2.1.1 View

4.2.1.1.1 Layout

WelcomeActivity: This class is the first screen that is shown to the user when the application is first launched. The user can then proceed to login or sign up to the system.

LoginActivity: This class is the representation of the screen where the user can enter their credentials and go into the system.

ForgotPasswordActivity: When the user forgets their password, they can use this screen to request a new password email from the system.

GuestActivity: This screen holds 2 pages for the guest user to sign up to the system, and 1 page for them to see the listed books.

SignUpActivity: This screen is for the new users to create an account by giving personal information.

MainActivity: This screen holds the suggestion list, match list and the listed books. For simplicity, these pages are designed to be horizontally scrollable.

ChatListActivity: This screen shows the ongoing and past chats of the user from where they can proceed to continue or start a chat.

ChatActivity: The user and a matched user can chat from this screen thanks to a quite familiar and simple massaging layout.

MatchListFragment: This segment shows the matches that the user is involved with.

SuggestionListFragment: This segment shows the suggested listed books to the user. From here, the user can agree or disagree with a suggestion.

PreferencesActivity: This screen is a navigation list from where the user can proceed to change several kinds of preferences which can be seen in the following sections.

HistoryActivity: This screen shows a list of transactions that the user has made recently.

ChatSettingsActivity: Here, the user can block or unblock a user with whom they have chatted recently.

TradeListActivity: The user can add and remove books that they are willing to use as a trade material in this section.

AccountSettingsActivity: In this section, the user can change personal and account related information.

WishlistActivity: The user sees and manipulates their wished books in this screen.

GeneralListFragment: This section shows the currently listed books in the system regardless of the user's preferences and likings.

BookInfoActivity: This screen shows information and a brief description of a particular book.

4.2.1.2 Controller

BookClubAPI: This class is meant to help the view package. It sends requests to the server and receives responses from the server via RESTful API over HTTP and keeps the session credentials statically. Its methods can give feedback if the process is done or return the data got from the server in a proper format on the client-side. Requests are recieved as JSON strings and parsed into the format that the view classes utilize.

4.2.1.3 Model

User: This class is an object to hold the user's personal info and account related data.

Book: This class is an object to hold a particular book's information.

4.2.2 Server Package

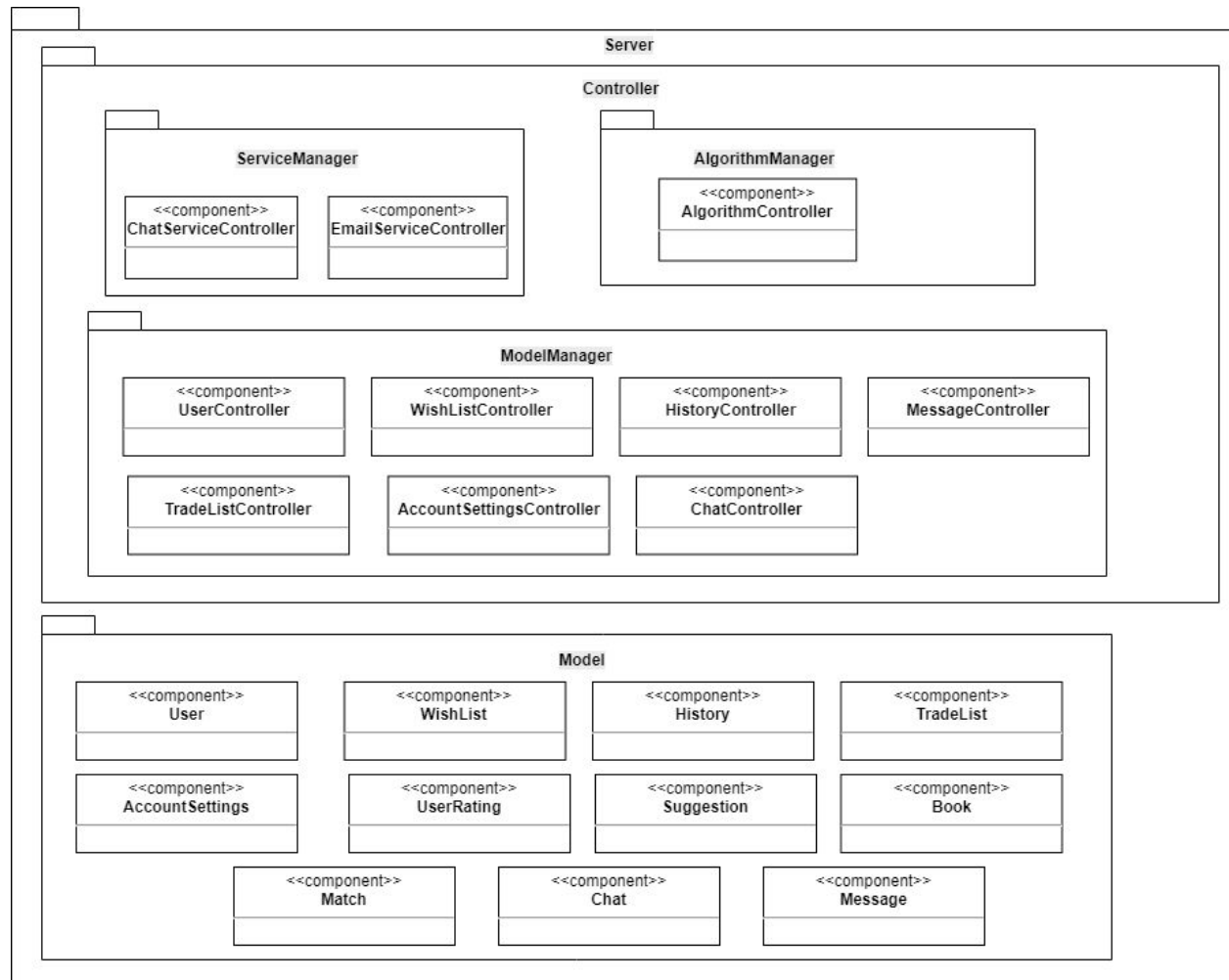


Figure 2: Package Diagram of the Server

The responsibility of the server side of the system is to handle different clients, get requests from them, keep data of BookClub users and the application, and manipulate those data by using different components in the server. Server package will be implemented in Python Django on a different machine (a server) to execute the whole package and keep the data of the application.

The server-side of the system is mainly grouped into two sections as model and controller packages. Controller package itself is divided into three main packages such

as AlgorithmManager, ServiceManager and ModelManager. Model package includes the classes dealing with the database in it. These packages are deeply analyzed in this section.

4.2.2.1 Model

Model part of the server is responsible for migrating the database to the application as a class. This way, the controllers will easily modify them within the server. These classes are User, UserRating, Book, Chat, Message, History, WishList, AccountSettings, TradeList, Match and Suggestion classes. These classes do not do anything except being a class of a particular table. In other words, these classes will basically be the attributes of a table (columns) in the server.

4.2.2.2 Controller

Controller part of the server is responsible for coordinating the user input between Model and View parts of the application.

4.2.2.2.1 ServiceManager

This package is a small-service package that includes in itself the classes that handle small services within the server-side. These classes are EmailServiceController and ChatServiceController.

EmailServiceController: This class handles the email service. In other words, when the application needs to notify a user with email (in signup or forgot password cases), this class configures the relevant SMTP ports and sends an email to the user effortlessly.

ChatServiceController: This class handles a chat between two users after the match is confirmed from both sides.

4.2.2.2.2 AlgorithmManager

This package is responsible for managing the execution of the core algorithm of BookClub. It performs this task by updating match lists and suggestion lists of users according to the results of the algorithm. It contains classes MatchListController and SuggestionListController.

MatchListController: This class is responsible for finding new matches for a user and updating the MatchList of a user accordingly. It does this by running the special match algorithm of BookClub.

SuggestionListController: This class is responsible for creating book suggestions for a user and updating the SuggestionList of a user accordingly. Book suggestions creating with a special recommendation algorithm.

4.2.2.2.3 ModelManager

This package is basically a controller package for all the classes of the model package. It has several controllers that handle the API requests of the particular model classes. Since all of them do the same thing, their main goal will be generally explained in this section. These controllers handle main three types of requests (some of the controllers have exceptions, the details are given in the further sections). These requests are POST, GET and DELETE.

1. When the request is POST, the controller (for example) signs a user up or adds a new item to the database. Thus, POST request is basically for adding or updating the items in the database.
2. When the request is DELETE, the controller deletes an item from the database. For example, when the user wants to delete a book from the wishlist/tradelist or wants to delete a particular conversation from the chat list, this request is used.
3. When the request is GET, the controller returns a data with the information from the database. It is almost the most used request in the application. For example, when the user wants to see a book's information the GET request is sent to the controller and the information of the book is returned as the JSON data.

4.3 Class Interfaces

4.3.1 Client

Please note that the class interfaces do not include getter and setter methods since they all will be implemented by default for each class. Also as each object will have a default

constructor, for the sake of simplicity across the report layout, these constructors will not be included inside the class interfaces. However, constructors with different than default parameters will be included.

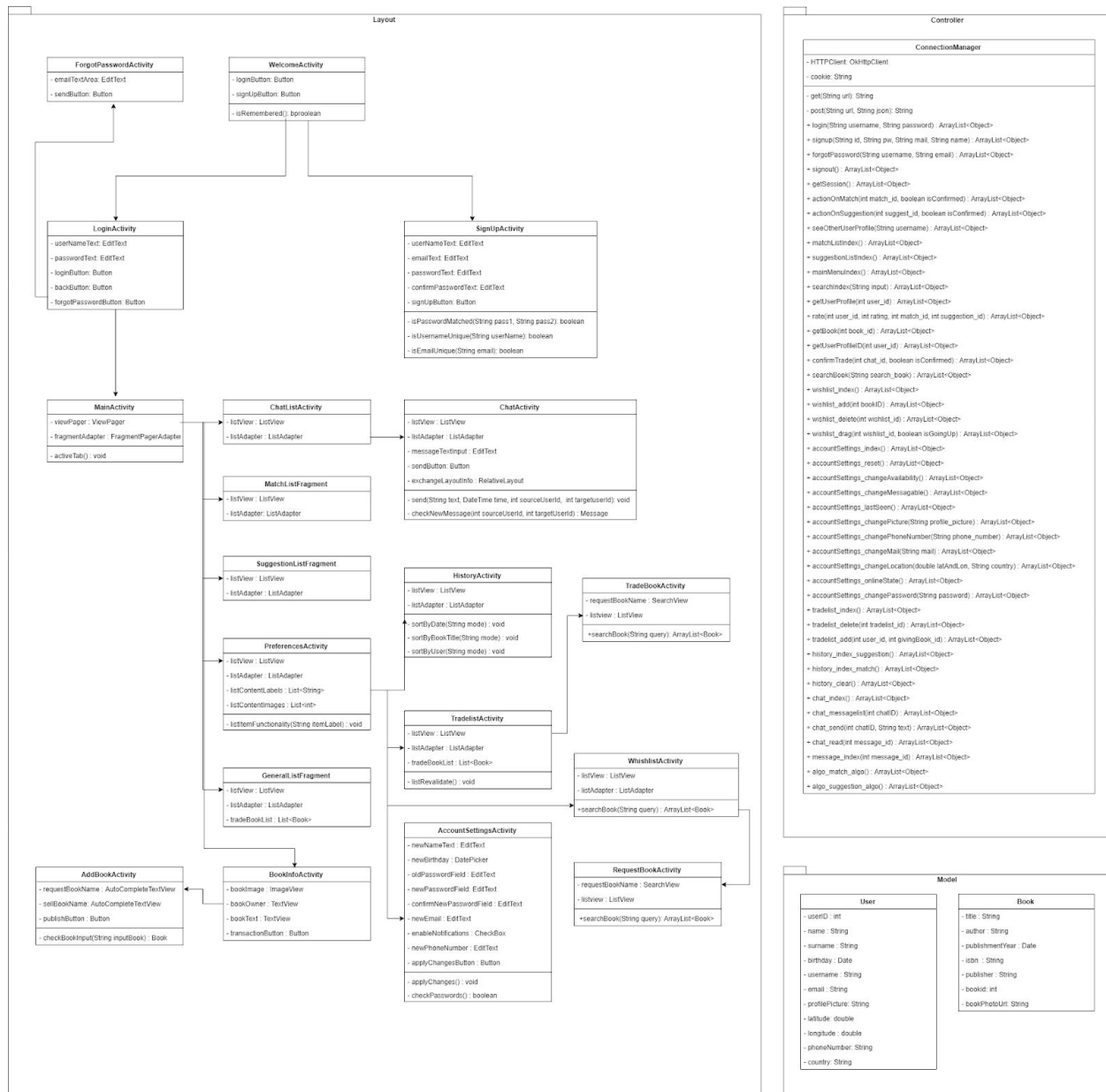


Figure : Class diagram of Client-Side

4.3.1.1 View

4.3.1.1.1 Layout Package

Please note that each activity has an onCreate() method by default and neither of the activities mentioned in the following section will contain an explanation regarding this method. The arguments of the onCreate() methods will not be changed. Also, the functionalities of the screen items (views) will be given inside the onCreate() methods using anonymous inner classes.

4.3.1.1.1.1 WelcomeActivity Class

| |
|--|
| Class: WelcomeActivity |
| This activity is the Java code of the welcome screen, working along with the XML file welcome_activity.xml. This activity is the Android manifest's main activity. |
| Attributes: |
| private Button loginButton : object of the button which brings login screen. private Button signUpButton : the object of the button which brings the signup page. |
| Functions: |
| private boolean isRemembered() : gathers the user credentials from the device and automatically authenticates if it finds valid credentials. |

4.3.1.1.1.2 LoginActivity Class

| |
|---|
| Class: LoginActivity |
| This activity is the Java code of the login screen, working along with the XML file login_activity.xml. |
| Attributes: |

private EditText userNameText : takes the username as input

private EditText passwordText : takes password as input while showing dots only for the characters.

private Button loginButton : starts the authentication process.

private Button backButton : takes the screen back to welcome screen.

private Button forgotPasswordButton : brings the forgot password screen.

4.3.1.1.1.3 ForgotPasswordActivity Class

Class: ForgotPasswordActivity

This activity is the Java code of the forgot password screen, working along with the XML file forgot_password_activity.xml.

Attributes:

private EditText emailTextArea : takes input for the email address that the system will send a new password request mail.

private Button sendButton : starts the password change procedure.

4.3.1.1.1.4 TradeBookActivity Class

Class: TradeBookActivity

This activity is the Java code of the trade book insertion screen, working along with the XML file trade_book_activity.xml.

Attributes:

private SearchView requestBookName: takes input for the search query.

private ListView listview: shows the list of search results

4.3.1.1.1.5 SignUpActivity Class

| |
|---|
| Class: SignUpActivity |
| This activity is the Java code of the signup screen, working along with the XML file sign_up_activity.xml. |
| Attributes: |
| private EditText userNameText : takes input for the username. private EditText emailText : takes input for the email address. private EditText passwordText : takes password input. private EditText confirmPasswordText : takes the confirming password input. private Button signUpButton : starts the signup procedure. |
| Functions: |
| private boolean isPasswordMatched(String pass1, String pass2) : every time the user types in a character for the confirming password, this method controls if two password inputs match. private boolean isUsernameUnique(String userName) : every time the user types in a character for the username, this function checks if the current username is unique. private boolean isEmailUnique(String email) : every time the user types in a character for the email address, this function checks if the current entry is unique. |

4.3.1.1.1.6 MainActivity Class

| |
|--|
| Class: MainActivity |
| This activity is the Java code of the main screen, working along with the XML file main_activity.xml. Note that this activity is not the main activity in terms of Android |

| |
|--|
| manifest entries. |
| Attributes: |
| <p>private ViewPager viewPager : holds the swipable fragments that are general list, suggestion list and match list.</p> <p>private FragmentPagerAdapter fragmentAdapter : adapts the fragments into the ViewPager object.</p> |
| Functions: |
| <p>private void activeTab(): changes the currently shown fragment. Called inside the FragmentPagerAdapter.</p> |

4.3.1.1.1.7 ChatListActivity Class

| |
|--|
| Class: ChatListActivity |
| This activity is the Java code of the chat list screen, working along with the XML file chat_list_activity.xml. |
| Attributes: |
| <p>private ListView listView : scrollable list that contains the chats of the user.</p> <p>private ListAdapter listadapter : adapts the data of the list items into the list item views and gives them particular functionalities.</p> |

4.3.1.1.1.8 ChatActivity Class

| |
|---|
| Class: ChatActivity |
| This activity is the Java code of the chat screen, working along with the XML file chat_activity.xml. |

| |
|---|
| Attributes: |
| <p>private ListView listView : scrollable list that holds the messages</p> <p>private ListAdapter listAdapter: adapts the messages into the list view.</p> <p>private EditText messageTextInput: takes message input.</p> <p>private RelativeLayout exchangeLayoutInfo : holds the content of the chat's subject such as the books that are being traded.</p> |
| Functions: |
| <p>private void send(String text, DateTime time, int sourceUserId, int targetUserId) : sends the message to the user.</p> <p>private Message checkNewMessages(int sourceUserId, int targetUserId) : gathers new messages from the server. It is the main function of a worker thread.</p> |

4.3.1.1.1.9 MatchListFragment Class

| |
|--|
| Class: MatchListFragment |
| This fragment is the Java code of the match list part of the scrollable view pager, working along with the XML file match_list_fragment.xml. |
| Attributes: |
| <p>private ListView listView : scrollable list that holds the matches.</p> <p>private ListAdapter listAdapter: adapts the match list items into the list view.</p> |

4.3.1.1.1.10 SuggestionListFragment Class

| |
|--|
| Class: SuggestionListFragment |
| This fragment is the Java code of the suggestion list part of the scrollable view pager, working along with the XML file suggestion_list_fragment.xml. |

| |
|---|
| Attributes: |
| private ListView listView : scrollable list that holds the suggestions. private ListAdapter listAdapter : adapts the suggestion list items into the list view. |

4.3.1.1.1.11 PreferencesActivity Class

| |
|---|
| Class: PreferencesActivity |
| This activity is the Java code of the preferences screen, working along with the XML file preferences_activity.xml. |
| Attributes: |
| private ListView listView : scrollable list that holds the preference tab items. private ListAdapter listAdapter : adapts the preference list items into the list view. private List<String> listContentLabel : contains the list item labels of the preferences list. private List<int> listContentImages : contains the integer representations of the icons of the list images. |
| Functions: |
| private void listItemFunctionality(String itemLabel) : uses intent to change screen based on which item is pressed. |

4.3.1.1.1.12 HistoryActivity Class

| |
|---|
| Class: HistoryActivity |
| This activity is the Java code of the history screen, working along with the XML file history_activity.xml. |

| |
|--|
| Attributes: |
| <p>private ListView listView : scrollable list that holds the history.</p> <p>private ListAdapter listAdapter: adapts the history list items into the list view.</p> |
| Functions: |
| <p>private void sortByDate(String mode) : sorts the history list based on their dates and the parameter mode which can either be “ascending” or “descending”.</p> <p>private void sortByBookTitle(String mode) : sorts the history list based on their book titles and the parameter mode which can either be “ascending” or “descending”.</p> <p>private void sortByUser(String mode) : sorts the history list based on their user names and the parameter mode which can either be “ascending” or “descending”.</p> |

4.3.1.1.13 RequestBookActivity Class

| |
|--|
| Class: RequestBookActivity |
| This activity is the Java code of the request book screen, working along with the XML file request_book_activity.xml. |
| Attributes: |
| <p>private ListView listView : scrollable list that holds the history.</p> |
| Functions: |
| <p>private void sortByDate(String mode) : sorts the history list based on their dates and the parameter mode which can either be “ascending” or “descending”.</p> <p>private void sortByBookTitle(String mode) : sorts the history list based on their book titles and the parameter mode which can either be “ascending” or “descending”.</p> <p>private void sortByUser(String mode) : sorts the history list based on their user names and the parameter mode which can either be “ascending” or “descending”.</p> |

4.3.1.1.1.14 TradeListActivity Class

| |
|--|
| Class: TradeListActivity |
| This activity is the Java code of the trade list screen, working along with the XML file trade_list_activity.xml. |
| Attributes: |
| private ListView listView : scrollable list that holds the tradable books. private ListAdapter listAdapter : adapts the tradable books' list items into the list view. private List<Book> tradeBookList : holds the tradable books. |
| Functions: |
| private void listRevalidate() : reconstructs the tradable book list after adding a new book to the list. |

4.3.1.1.1.15 AccountSettingsActivity Class

| |
|--|
| Class: AccountSettingsActivity |
| This activity is the Java code of the account settings screen, working along with the XML file account_settings_activity.xml. |
| Attributes: |
| private EditText newNameText : input for the personal name. private DatePicker newBirthday : date picker to select a new birthday. private EditText oldPasswordField : input for the current password if the user wants to change the password. private EditText newPasswordField : input for the new password if the user wants to |

change the current password.

private EditText confirmPasswordField: input for the confirmed new password if the user wants to change the current password.

private EditText newEmail : input for new email address.

private CheckBox enableNotifications: check box to enable/disable the notifications from the application.

private EditText newPhoneNumber: input for the new phone number.

private Button applyChangesButton: triggers the changes.

Functions:

private void applyChanges(): sends a series of queries to make all the requested changes.

private boolean checkPasswords(): checks if the passwords match.

4.3.1.1.16 WishlistActivity Class

Class: WishlistActivity

This activity is the Java code of the wish list screen, working along with the XML file wishlist_activity.xml.

Attributes:

private ListView listView: scrollable list that holds the wished books.

private ListAdapter listAdapter: adapts the wished books' list items into the list view.

Functions:

private void listRevalidate(): reconstructs the tradable book list after adding a new book or removing a book from the list.

4.3.1.1.1.17 GeneralListFragment Class

| |
|--|
| Class: GeneralListFragment |
| This fragment is the Java code of the listed books' list, working along with the XML file general_list_fragment.xml. |
| Attributes: |
| private ListView listView: scrollable list that holds the listed books. private ListAdapter listAdapter: adapts the listed books' list items into the list view. private List<Books> tradeBookList: list of the listed books to hold in memory. |

4.3.1.1.1.18 BookInfoActivity Class

| |
|--|
| Class: BookInfoActivity |
| This activity is the Java code of the book information screen, working along with the XML file book_info_activity.xml. |
| Attributes: |
| private ImageView bookImage : image of the book. private TextView bookOwner : text of the owner of the book. private TextView bookText : description of the book. private Button transactionButton : starts the indicated transaction when pressed. |

4.3.1.2 Controller

4.3.1.2.1 BookClubAPI Class

Following funtions in the BookClubAPI class has always a return attribute in the ArrayList. At 0 index, the status of the request returns (error or success). At index 1, the

message coming from the server returns. Finally at index 2, the result of the function returns in a specific format whose explanation can be found in the following table. The functions with parameters send POST request. The ones without parameters send GET request.

| |
|---|
| Class: BookClubAPI |
| This class is used to send a request to the server or receive a result from the server over RESTful API. Also, it puts the taken JSON results into the required object format in the client. |
| Attributes: |
| <p>private OkHttpClient HTTPClient: stores the data if there is an active connection.</p> <p>private String cookie: stores the token that is got from the server and used to verify that the current connection is valid. To achieve it, the user needs to login.</p> |
| Functions: |
| <p>private get(String url): String: sends a GET request to the server. If the user is authenticated, the method receives and sets the cookie.</p> <p>private post(String url, String json): String: sends a POST request to the server by parsing the given string into JSON. If the user is authenticated, the method receives and sets the cookie.</p> <p>public login(String username, String password) : ArrayList<Object>: If login is successful, sets the session cookie.</p> <p>public signup(String id, String pw, String mail, String name) : ArrayList<Object>: Only returns the success state.</p> <p>public forgotPassword(String username, String email) : ArrayList<Object>: returns the new password.</p> <p>public signout() : ArrayList<Object>: signs out from the server and destroys the</p> |

cookie.

public getSession() : ArrayList<Object>: returns the user id in the session.

public actionOnMatch(int match_id, boolean isConfirmed) : ArrayList<Object>: sends the match id and a boolean that indicates if the match is confirmed.

public actionOnSuggestion(int suggest_id, boolean isConfirmed) : ArrayList<Object>: sends the suggestion id and the boolean that indicates if the suggestion is confirmed.

public seeOtherUserProfile(String username) : ArrayList<Object>: returns the user object with a given username.

public matchListIndex() : ArrayList<Object>: gets the matchlist of the user logged in with the book objects.

public suggestionListIndex() : ArrayList<Object>: gets the suggestlistof the user logged in with the book objects.

public mainMenuIndex() : ArrayList<Object>: gets the main menu book lists with the book objects.

public searchIndex(String input) : ArrayList<Object>: returns the book objects that matches with the specified keyword.

public getUserProfile(int user_id) : ArrayList<Object>: returns a user object with the given user id

public rate(int user_id, int rating, int match_id, int suggestion_id) : ArrayList<Object>: sends a POST request to rate specified user by giving the rate points.

public getBook(int book_id) : ArrayList<Object>: returns the book with the given book id.

public getUserProfileID(int user_id) : ArrayList<Object>: returns a user object with the given user id.

public confirmTrade(int chat_id, boolean isConfirmed) : ArrayList<Object>: confirms or rejects the trade by giving the chat id of the trade.

public searchBook(String search_book) : ArrayList<Object>: returns the book

objects that matches with the specified keyword.

public wishlist_index() : ArrayList<Object>: gets the book objects in the wishlist of the user logged in.

public wishlist_add(int bookID) : ArrayList<Object>: adds the book with the given book id into the wishlist.

public wishlist_delete(int wishlist_id) : ArrayList<Object>: deletes the specified row in the wishlist by giving its id.

public wishlist_drag(int wishlist_id, boolean isGoingUp) : ArrayList<Object>: sends a request to change the order of the book in the wishlist.

public accountSettings_index() : ArrayList<Object>: gets all the options of the account settings of the user logged in.

public accountSettings_reset() : ArrayList<Object>: resets all the options to the defaults.

public accountSettings_changeAvailability() : ArrayList<Object>: toggles the availability option in the account settings.

public accountSettings_changeMessagable() : ArrayList<Object>: toggles the messagable option of the user logged in.

public accountSettings_lastSeen() : ArrayList<Object>: returns the last seen information.

public accountSettings_changePicture(String profile_picture) : ArrayList<Object>: changes the profile picture with the given input.

public accountSettings_changePhoneNumber(String phone_number) : ArrayList<Object>: changes the phone number with the new one.

public accountSettings_changeMail(String mail) : ArrayList<Object>: changes the mail address with the new one.

public accountSettings_changeLocation(double latAndLon, String country) : ArrayList<Object>: changes the user location with the given coordinates.

public accountSettings_onlineState() : ArrayList<Object>: toggles the online state.

public accountSettings_changePassword(String password) : ArrayList<Object>:

changes the password with the given password.

public tradelist_index() : ArrayList<Object>: returns the book objects in the trade list.

public tradelist_delete(int tradelist_id) : ArrayList<Object>: deletes the row of the tradelist with the given id.

public tradelist_add(int user_id, int givingBook_id) : ArrayList<Object>: adds a new book with given id into the trade list.

public history_index_suggestion() : ArrayList<Object>: returns the book objects in the suggestion history.

public history_index_match() : ArrayList<Object>: returns the book objects in the match history.

public history_clear() : ArrayList<Object>: clears all the history information.

public chat_index() : ArrayList<Object>: returns the list of chats.

public chat_messagelist(int chatID) : ArrayList<Object>: returns all the messages in the specified chat.

public chat_send(int chatID, String text) : ArrayList<Object>: pushes a text into the specified chat.

public chat_read(int message_id) : ArrayList<Object>: changes the state of the given message into read.

public message_index(int message_id) : ArrayList<Object>: returns the message information.

public algo_match_algo() : ArrayList<Object>: indicates that the user wants the match algorithm to re-run.

public algo_suggestion_algo() : ArrayList<Object>: indicates that the user wants the match algorithm to re-run.

4.3.1.3 Model

4.3.1.3.1 Book Class

| |
|---|
| Class: Book |
| This class is used to store and manipulate the information of a book in the system. Defining all the variables is not a must |
| Attributes: |
| private String title: stores the title of the book private String author: stores the author of the book private Date publishmentYear: stores the publish year of the book private String isbn: stores the isbn of the book private String publisher: stores the publisher of the book private int bookid: stores the id number of the book private String bookPhotoUrl: stores the cover image of the book |

4.3.1.3.2 User Class

| |
|---|
| Class: User |
| This class is used to store and manipulate the information of a user in the system. Defining all the variables is not a must. |
| Attributes: |
| private int userID: stores the ID number of the user private String name: stores the name of the user private String surname: stores the surname of the user private Date birthday: stores the birthday of the user private String username: stores the username of the user |

private String email: stores the email address of the user

private String profilePicture: stores the profile picture of the user

private double latitude: stores the latitude of where the user lives

private double longitude : stores the longitude of where the user lives

private String phoneNumber: stores the phone number of the user.

private String country: stores the country of the user.

4.3.2 Server

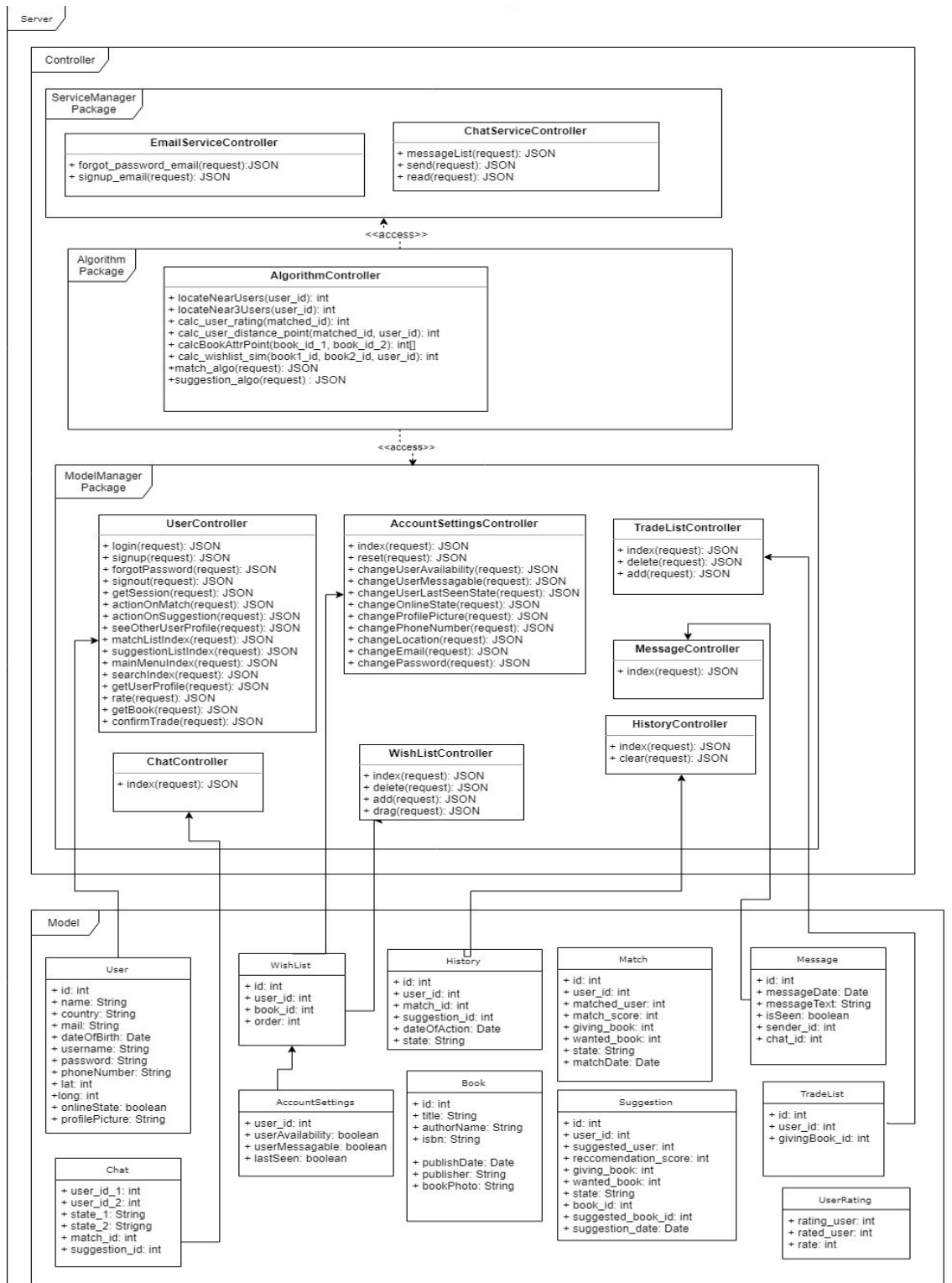


Figure : Class diagram of Server-Side

4.3.2.1 Controller

4.3.2.1.1 AlgorithmManager Package

4.3.2.1.1.1 AlgorithmController Class

| |
|---|
| Class: AlgorithmController |
| This class is used to create and update the match and suggestion lists of users. |
| Functions: |
| suggestion_algorithm(request): This function creates and updates the suggestion list of a user by creating new suggestions according to some factors like user's previous activity etc. Detailed explanation of the algorithm can be found in Server Algorithm and Implementation section. |
| match_algorithm(request): This function creates and updates the match list of a user by creating new matches according to some factors like user's previous activity etc. Detailed explanation of the algorithm can be found in Server Algorithm and Implementation section. |

4.3.2.1.2 ServiceManager Package

4.3.2.1.2.1 EmailServiceController Class

| |
|--|
| Class: EmailServiceController |
| This class is used to control and manipulate the email sending functionality of BookClub. This is necessary to connect with users in specific cases such as when they forgot their password. |
| Functions: |
| forgot_password_email(request): This function gets the user from the session, sends a forgot password email to the user. |

signup_email(request): This function gets the user from the session and sends a signup email to the user.

4.3.2.1.2.2 ChatServiceController Class

Class: ChatServiceController

This class is used to control and manipulate the chat functionality of BookClub.

Functions:

createChatSession (user_id1, user_id2): This function creates a chat session between the users taken by parameters.

terminateChatSession (user_id1, user_id2): This function terminates a chat session when a trade is completed.

4.3.2.1.3 ModelManager Package

4.3.2.1.3.1 UserController Class

Class: UserController Class

This class is a User model's controller class

Functions:

getSession(request): This function gets the user in session

login(request): This function handles the user's login request

signup(request): This function handles the user's signup request

forgotPassword(request): This function handles the user's forgot password request

signout(request): This function handles the user's sign out request

seeOtherUserProfile(request): This function handles the request created when the user checks other user's profile

getUserProfile: This function returns the user information with the given id

matchlistIndex(request): This function returns the detailed data of the match list when the user wants to see it

suggestionlistIndex(request): This function returns the detailed data of the suggestion list when the user wants to see it

mainMenuIndex(request): This function returns the data of the main menu when the user refreshes it or clicks on the main menu. In the main menu, we show a list of random book trades.

searchIndex(request): This function returns the trades that include the book that user searched

searchBook(request): This function returns the book that includes the data that the user searched

getBook(request): This function returns the book information with the given id

actionOnMatch(request): This function handles the user's actions on matches such as confirming and rejecting

actionOnSuggestion(request): This function handles the user's actions on suggestions such as confirming and rejecting

confirmTrade(request): This function handles the user's actions on trades such as confirming and rejecting

rateUser(request): This function handles the request created when the user rates another user after a trade

4.3.2.1.3.2 WishListController Class

Class: WishListController Class

This class is a Wishlist model's controller class

Functions:

index(request): This function returns the detailed data of the wishlist for the user in session

delete(request): This function is activated when the user deletes a book from the wishlist

add(request): This function is activated when the user adds a new book to the wishlist

drag(request): This function is activated when the user reorders the wishlist books

4.3.2.1.3.3 HistoryController Class

Class: HistoryController Class

This class is a History model's controller class

Functions:

indexMatch(request): This function returns the detailed match history for the user in session

indexSuggestion(request): This function returns the detailed suggestion history for the user in session

clear(request): This function is activated when the user clears the whole history

4.3.2.1.3.4 TradeListController Class

Class: TradeListController Class

This class is a Tradelist model's controller class

Functions:

index(request): This function returns the detailed data of the trade list for the user in session

delete(request): This function is activated when the user deletes a trade instance from the trade list

add(request): This function is activated when the user adds a new trade instance to the trade list

4.3.2.1.3.5 AccountSettingsController Class

| |
|---|
| Class: AccountSettingsController Class |
| This class is a AccountSettings model's controller class |
| Functions: |
| <p>index(request): This function returns the data of the user's current privacy settings</p> <p>reset(request): This function is activated when the user resets the privacy settings</p> <p>changeUserAvailability(request): This function handles the user's change the availability request</p> <p>changeUserMessagable(request): This function handles the user's change messagability state request</p> <p>changeLastSeenState(request): This function handles the user's last seen state request</p> <p>changeOnlineState(request): This function handles the user's change online state request</p> <p>changeProfilePicture(request): This function handles the user's change profile picture request</p> <p>changePhoneNumber(request): This function handles the user's change phone number request</p> <p>changeLocation(request): This function handles the user's change location request</p> <p>changeEmail(request): This function handles the user's change email request +</p> <p>changePassword(request): This function handles the user's change password</p> |

| |
|---------|
| request |
|---------|

4.3.2.1.3.6 ChatController Class

| |
|------------------------------------|
| Class: ChatController Class |
|------------------------------------|

| |
|---|
| This class is a Chat model's controller class |
|---|

| |
|-------------------|
| Functions: |
|-------------------|

| |
|---|
| index(request): This function returns the chat list of the user in session |
|---|

4.3.2.1.3.7 MessageController Class

| |
|---------------------------------------|
| Class: MessageController Class |
|---------------------------------------|

| |
|--|
| This class is a Message model's controller class |
|--|

| |
|-------------------|
| Functions: |
|-------------------|

| |
|---|
| index(request): This function returns the message list of the specific chat instance that user selected to see |
|---|

4.3.2.2 Model

4.3.2.2.1 User Class

| |
|--------------------------|
| Class: User Class |
|--------------------------|

| |
|--|
| This class is a model class for the user table from the database |
|--|

| |
|--------------------|
| Attributes: |
|--------------------|

| |
|---|
| id: This attribute represents the user id column |
|---|

name: This attribute represents the user's name column

country: This attribute represents the user's country column

mail: This attribute represents the user's mail column

phoneNumber: This attribute represents the user's phone number column

dateOfBirth: This attribute represents the user's date of birth column

username: This attribute represents the user's username column

password: This attribute represents the user's password column

lat: This attribute represents the user's latitude column. It is needed for the GPS purposes

long: This attribute represents the user's longitude column. It is needed for the GPS purposes

onlineState: This attribute is a boolean attribute and represents the user's online/offline state column

profilePicture: This attribute stores the name of the user's profile photo in the database

4.3.2.2.2 WishList Class

| |
|--|
| Class: WishList Class |
| This class is a model class for the wishlist table from the database |
| Attributes: |
| <p>id: This attribute represents the wishlist id column</p> <p>user_id: This attribute represents the user's id column inside the wishlist table as a foreign key</p> <p>book_id: This attribute represents the book's id column inside the wishlist table as a foreign key</p> |

4.3.2.2.3 History Class

| |
|--|
| Class: History Class |
| This class is a model class for the history table from the database |
| Attributes: |
| id: This attribute represents the history id column user_id: This attribute represents the user's id column inside the history table as a foreign key match_id: This attribute represents the match id column inside the history table as a foreign key. It can be null if it is a history data of a suggestion suggestion_id: This attribute represents the suggestion match transaction id column inside the history table as a foreign key. It can be null if it is a history data of a match dateOfAction: This attribute represents the date of the match state: This attribute represents the state(confirmed or rejected) of the match or suggestion |

4.3.2.2.4 TradeList Class

| |
|---|
| Class: TradeList Class |
| This class is a model class for the tradelist table from the database |
| Attributes: |
| id: This attribute represents the wishlist id column user_id: This attribute represents the user's id column inside the trade list table as a foreign key givingBook_id: This attribute represents the user's giving book's id column inside the trade list table as a foreign key |

4.3.2.2.5 AccountSettings Class

| |
|--|
| Class: AccountSettings Class |
| This class is a model class for the account table from the database |
| Attributes: |
| user_id: this attribute represents the user's id column inside the account table as a foreign key userAvailability: this attribute is a boolean attribute that will be turned on or off depending on the user's availability case userMessagable: this attribute is a boolean attribute that will be turned on or off depending on the user's chat availability state lastSeen: this attribute is a boolean attribute that will be turned on or off depending on the user's desire to showcase their last seen time or not. |

4.3.2.2.6 Chat Class

| |
|--|
| Class: Chat Class |
| This class is a model class for the chat table from the database |
| Attributes: |
| user_id_1: This attribute represents the first user's id in the chat table as a foreign key state_1: This attribute represents the first user's state (online or offline) user_id_2: this attribute represents the second user's id in the chat as a foreign key state_2: This attribute represents the second user's state (online or offline) match_id: This attribute shows the match_id if the chat is opened after the confirmation of a match suggestion_id: This attribute shows the suggestion_id if the chat is opened after the |

| |
|------------------------------|
| confirmation of a suggestion |
|------------------------------|

4.3.2.2.7 Message Class

| |
|---|
| Class: Message Class |
| This class is a model class for the message table from the database |
| Attributes: |
| id: This attribute represents the sent message's id in the message table messageText: This attribute represents the sent message's text in the message table messageDate: This attribute represents the sent message's date in the message table isSeen: This attribute is a boolean attribute and it represents the sent message's state; seen or not seen sender_id: This attribute represents the id of the user that sends the message shat_id: This attribute represent the chat_id that the message belong |

4.3.2.2.8 Book Class

| |
|---|
| Class: Book Class |
| This class is a model class for the book table from the database |
| Attributes: |
| id: This attribute represents the book id column title: This attribute represents the book's name column authorName: This attribute represents the book's author's name column isbn: This attribute represents the book's isbn column publisher: This attribute reperenst the book's publisher |

publishDate: This attribute represents the book's publish date column
bookPhoto: This attribute represents the book's image's name in the database

4.3.2.2.9 Match Class

| |
|---|
| Class: Match Class |
| This class is a model class for the match table from the database |
| Attributes: |
| id: This attribute represents the match id column user_id: This attribute represents the user's id column matched_id: This attribute represents the matched user's id column match_score: This attribute represents the match score of the match instance giving_book: This attribute represents giving book's id column wanted_book: This attribute represents the wanted book's id column state: 'confirmed', 'rejected', 'nothing' matchDate: This attribute represents the date of the match |

4.3.2.2.10 Suggestion Class

| |
|--|
| Class: Suggestion Class |
| This class is a model class for the suggestion table from the database |
| Attributes: |
| id: This attribute represents the suggestion id column user_id: This attribute represents the user's id column suggested_user: This attribute represents the suggested user's id column |

recommendation_score: This attribute represents the recommendation score of the suggestion instance

giving_book: This attribute represents book_id that user will give

wanted_book: This attribute represents the book_id that is user wishlist

state: 'confirmed', 'rejected', 'nothing'

suggested_book_id: This attribute represent the book_id that is suggested to the user

suggestionDate: This attribute represents the date of the suggestion

5. Impact of Engineering Solutions

5.1. Economical Impacts

BookClub may have economic impacts on Book Sale Marketplaces. BookClub gives its users an opportunity of trading their books without paying rather than buying new books. Since it is a much more cost efficient way, many people prefer trading. This may lead to a decrease in the sales of books. Additionally, the budget that people reserve to buy books can decrease. For instance, in universities, most of the course books are expensive, thus, with BookClub students can spend their limited budget to other things.

5.2. Environmental Impacts

If BookClub becomes popular and trading of used books are preferred instead of buying new books, the book sales can be decreased. Due to this decreasing, publishers may reduce the number of prints of the new books.

Additionally, BookClub tries to decrease book blackmarketing, which unfortunately enlarges its sales day by day. BookClub's popularity could definitely reduce and even stop the blackmarket in some areas. It would be a huge environmental impact, since the number of trees cut down for the blackmarketing could decrease drastically.

5.3. Social Impacts

- BookClub aims to increase the reading amount of the society by providing them an easy-to-use channel for reaching out new books.
- BookClub also aims to reduce book piracy and black market sales by presenting a practical book trading platform. The users of BookClub will be able to obtain a lot of original books without paying any price and by this way, book piracy and buying books from black market will become less advantageous for readers.
- BookClub will create a community of book lovers. For the users, it will be possible to meet new people with similar taste of books.
- In BookClub, the users will be able to rate each other after both parties confirm that trade was processed. By this way, the user ratings will be accurate values that demonstrate the satisfaction of the users from trades with a particular user. This will increase the trustworthiness of the app.

6. Tools and Technologies Used

6.1. Development Tools

6.1.1 Pycharm

PyCharm was used as an editor, mainly by the server team. Since PyCharm had great tools, debuggers and support plugins for Python 3 and Django Framework, we decided to use it right away. Additionally, the team was already familiar with PyCharm, because it has a similar and easy interface as in IntelliJ, which everyone knew from CS 319 - Object-Oriented Software Engineering course.

6.1.2. Visual Studio Code

Visual Studio Code is another editor that some members of the server team used during the implementation. The reason why some members used VScode was mainly its weight. In comparison to PyCharm, VScode is much lighter and simpler. Since some

laptops had a difficulty to run PyCharm everytime, the members decided to switch to VScode. There was no problem in combining codes or merging them, thus we decided to go on with two editors in the server team.

6.1.3. GitHub

GitHub was the main tool that we used for combining, updating and merging codes. We used GitHub Desktop for updating the codes from the branches. Each team (server and client) had their own branches and additionally, each member had their own branch. We also had a master branch for the final code. This way, we ensured that there was no problem in merging the code. If someone wanted to change their code they would do it in their own branch. Later, after the detailed code review and consensus we were updating the main branches which are server and client.

6.1.4. Postman

Since BookClub is a client-server application, we needed a tool for communicating and testing our RestAPI. Postman was used as a communication and testing tool. Its simple UI and support for easily adjusting the requests was very helpful for both client and server team. [8]

6.1.5. Xampp

Since it is not always easy to find a good external server, we used our local server for testing our own codes in each laptop. Since all of the members were familiar with Xampp from CS 353 - Database Systems course, we decided to use this tool. We used Xampp for running the local server in the laptops. Additionally, we used Xampp for running a public local server so that we could access the common server. It saved us a lot of time and it was on budget. [9]

6.1.6. phpMyAdmin

BookClub required a good and easily accessible database that supports MySQL, we decided to stick to the good old phpMyAdmin, which is normally supported by Xampp.

Each member had a database in their local. Before switching to the common database, phpMyAdmin was very useful for the individual implementation.

6.1.7. HeidiSQL

We began using HeidiSQL after switching to the common database. As it is seen from the tool's name, it has the same function as phpMyAdmin. However, HeidiSQL has much better interface and more options than phpMyAdmin. For example, it was easier to change the foreign key restrictions in HeidiSQL than in the other one. It also had an advantage of displaying the local database. [10]

6.1.8. Trello

BookClub is a huge application with a lot of functions both in client and in the server side. It would be nearly impossible for our team to understand all requirements correctly, remember all “todo”s and add new tasks in real life. Trello is a very simple and user-friendly project management tool that we used throughout the whole project. We had four branches in Trello, which are API, Algorithm, Server and Client. API was a common branch where the server side was updating the requests and responses for the client side to use easily. Server and Client branches were being used by each teams for adding new tasks. Algorithm was used by server team for adding the tasks for the algorithm of the BookClub only. Overall, Trello was very useful in the BookClub implementation progress from the very start of the application.

6.1.9. Android Studio

Android Studio 3.0 was used as the development environment of the user interface code. As Android Studio is a customized version of Netbeans IntelliJ Idea, it has some excellent helping features, such as code completion, automatic constructor, getter, setter generation, partial compilation, UI design tools, etc. Also, thanks to the maven it contains, it is very easy to install and use other libraries and plugins.

6.1.10. Genymotion Emulator

Genymotion Emulator is a third party Android virtual machine program which can be integrated with Android Studio. As Android Studio's own virtual machine is slow and unresponsive, Genymotion became a must for the development process of the user interface. It also has some needed features such as fake GPS location and battery usage monitoring. [11]

6.2. Libraries and Frameworks Resources

6.2.1 Square Picasso

Picasso is a third party library that eases the process of loading images from URL and adapting them into the respective container. It, by default, has a very efficient caching mechanism that stops the program from loading an image from the same URL more than once. Also, it handles all the operations in asynchronous tasks, therefore, it rarely uses the UI thread and slows the application down. [12]

6.2.2 Swipe to Dismiss Undo

This third party library eases the operation of swiping items from a list view. It is not very easy to use this library as it requires many adjustments in the code to get working, however, it has an elegant approach and it does what it provides to do. [13]

6.2.3 Spots Dialog

This is yet another third party library that provides an elegant AlertDialog component for the Android ecosystem. It is used in every place where the user is asked to wait until the network access is done. As ProgressDialog component became deprecated by Android API 26, the only way of creating a progress dialog was to recreate an AlertDialog component with its own view holder. Thanks to Spots, it became very easy to create the wanted dialogs. [14]

6.2.4 LovelyDialog

Similar to Spots dialog, LovelyDialog is an AlertDialog, except that is much more customizable and it supports custom actions. Dialogs are the easiest ways of getting inputs from users therefore their appearance and responsiveness is very crucial for an app. Thanks to this library, we can show very good looking dialogs. [15]

6.2.5 Pandas

In server side we use pandas dataframes and its methods to hold, iterate and modify our data. In our database we have many rows in our tables (users, books, wish lists, tradelists and etc). In some algorithms such as match and suggestion algorithm we need to iterate over these tables for several users. Since, pandas dataframe is created to ease and fasten the modification and iteration in large datasets, it is very useful for our application. [16]

6.2.6 Factory-boy and Faker

We used these libraries to generate data for our Django Models. Since we need data in our Database to control and test our program, using this library helped us a lot. With the help of these libraries we could easily seed the tables in the database. [17]

6.2.7 Django

Django is a very powerful Python Framework that we used to implement the server part of the BookClub. Django comes with a good folder structures where we can add a new apps to the application, control the configurations, databases and settings easily. [18]

6.2.8 Django-cors-headers

Django-cors-headers library was added to the server part of the application, after having conflicts of allocating the headers from the client side. In other words, when the client side was sending requests to the server, they were getting errors such as 'the header is not allocated correctly' which meant they had to implement the header manually. This

would be a waste of time, since the codes were constantly changing, thus Django-cors-headers came very handy in automatically adjusting the headers for the client side. [19]

6.2.9 djangoestframework

Djangoestframework is a library for Django Python Framework which creates a plugin to implement the RestAPI. We used this framework for creating routes with GET, POST and DELETE gates. [20]

6.2.10 mysqlclient

BookClub has a huge database with MySQL support. Since Django Python Framework had a sqlite support only, we needed to add mysql plugin in order to adjust it to the MySQL database. Mysqlclient was installed and configured to the application for creating this support.

6.2.11 Android SDK 28

As BookClub is an Android application, it was coded in Android SDK. We kept BookClub as up-to-date as possible therefore we used the newest Android SDK, however it supports Android devices that support SDK 26 and above. As SDK 28 is the most recent version, we used the most recent technologies that it provides such as easier asynchronous task creation.

7. Engineering Solutions and Contemporary Issues

7.1 Internet Related Issues

BookClub requires internet connection to perform its functionalities. If users are connected to internet, then they will be able to receive matches and suggestions, send messages in chat, search for books, etc. However, if they are not connected to internet, then they will not be able to use BookClub's functionalities. But we think that this will not

be a huge problem, since in the world of 2019, it is not that hard to reach internet connection. Almost every cafe and restaurant offers free Wi-Fi and most people have cellular network functionality in their mobile phones.

7.2 Issues Related to Huge Market Size

In our modern world, approximately 12 books are released in every hour in Amazon [21]. It means that around 288 new books are released in a day. In order to fulfill its functionalities successfully, the books dataset of BookClub should always be up to date. It is a difficult task to achieve, because it means that the books dataset will grow very large in time. Therefore, in the future versions of BookClub, it will be necessary to find a way to efficiently add and store the newly released books in the database.

7.3 Language and Communication Issues

The initial market of BookClub will be Turkey. However, we implemented BookClub in English. The reason is, our aim for BookClub is to make it easily adaptable to every country in the world. For this purpose, we decided that implementing our platform in English would be better since it can be considered as a universal language in today's world. Moreover, we believe there will be a lot of university students among the users and there are lots of international students who are studying in Turkey. We decided to implement BookClub in the light of these factors.

7.4 Data Privacy Issues

In the database of BookClub, we store some personal data of the users such as their phone numbers, email addresses, locations, dates of birth, etc. These data will not be shared with any third party people or company. We will keep the data of our users private and it will not be possible for any other third party people to reach our database. We also keep record of user actions as the user's history. These actions of users will also be kept private, and only the users themselves will be able to see their history. Users will only be able to see their own wishlists and tradelists. By this way, we aimed to

find matches and create suggestions for our users without making their personal data available to everyone, because otherwise, some users could make pressure to the user to keep a book for them and reject other matches/suggestions, which would not be a pleasant experience for the user.

7.5 Security Issues

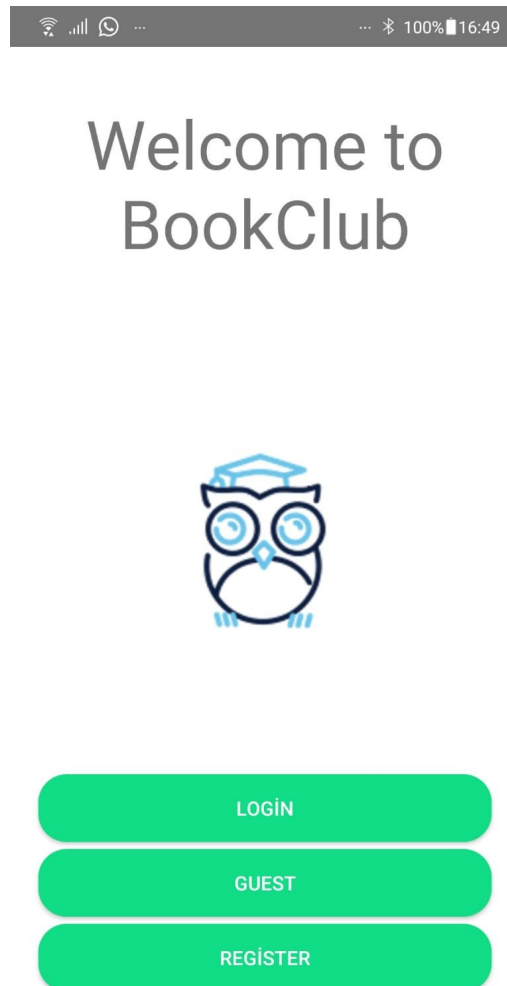
For BookClub, the security of the user data is very critical. In order to make sure that user accounts are secure, we hash the user passwords using md5() function with salting, then store the hashed passwords in our database during signup. md5 is a one-way (irreversible) hash function which basically maps an input with any size to a string of bits with a fixed size [22].

Salting is basically concatenating the password the user entered with a predefined string and hashing the resulting string. md5 with salting increases the security a lot, because a lot of md5 decrypters have been built and people with bad intentions can reach them easily. Decrypting passwords which were hashed using salting is much more harder than decrypting directly hashed passwords [23]. When users enter their password in the login screen, we hash these strings using salting again and compare the resulting string of bits with the password which is stored in our database. If they match, then the gates of BookClub are opened for the user. Otherwise, no luck.

This also prevents the developers and database maintainers to see user passwords, which is crucial in terms of user information security.

8.User Manual

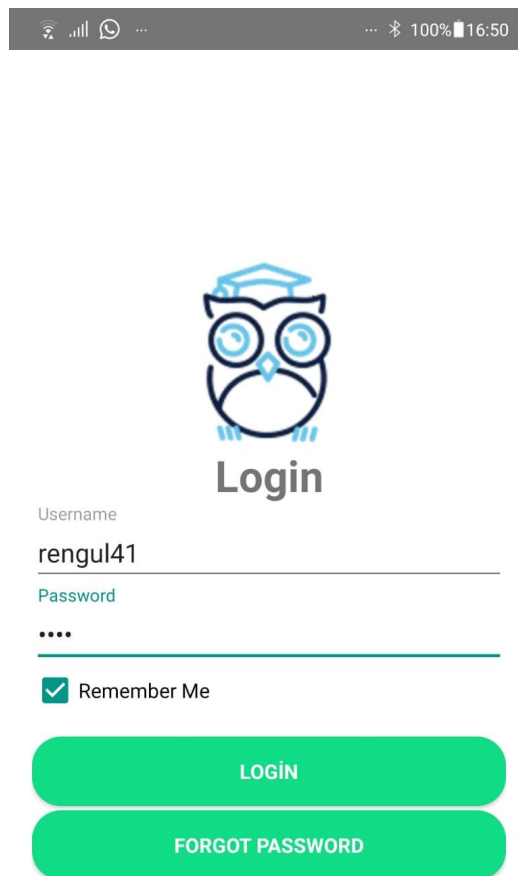
8.1 Initial Main Page



Welcome Screen

When the user first downloads BookClub and enters the application, he/she faces this screen. From this screen, the user can either login with the credentials (if they are already registered), login as a guest - without any credentials or register to the application. Each of these buttons leads to the separate screens.

8.2 Login Page for Normal Users



The image shows a mobile application login screen. At the top is a status bar with icons for signal, Wi-Fi, and battery, and the time 16:50. Below the status bar is a large, light blue owl logo wearing a graduation cap. Under the logo, the word "Login" is written in a bold, black, sans-serif font. Below "Login" are two input fields. The first is labeled "Username" in a small, grey font, and contains the text "rengul41". The second is labeled "Password" in a small, grey font, and contains four black dots. Below the password field is a checkbox with a blue checkmark, followed by the text "Remember Me". At the bottom of the form are two large, rounded rectangular buttons. The top button is blue with the text "LOGIN" in white, uppercase letters. The bottom button is blue with the text "FORGOT PASSWORD" in white, uppercase letters.

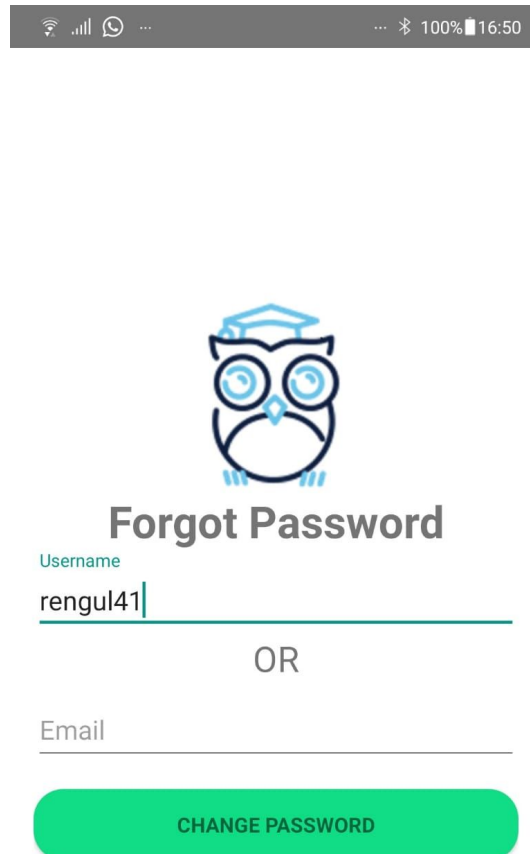
Login Screen

When the user presses the login button in Initial Main Page, they are lead to the Login Page for Normal Users screen. In this screen, the user enters their username and password and presses the login button. If the credentials are correct, they are lead to the General Page. Else, they get the error message which states that the entered credentials are incorrect.

The user can also press the remember me checkbox in order to automatically enter the app every time they click on it.

Additionally, if the users forget their password they can press the forgot password button, which will lead them to the Forgot Password Page.

8.3 Forgot Password Page



Forgot Password

Username

rengul41

OR

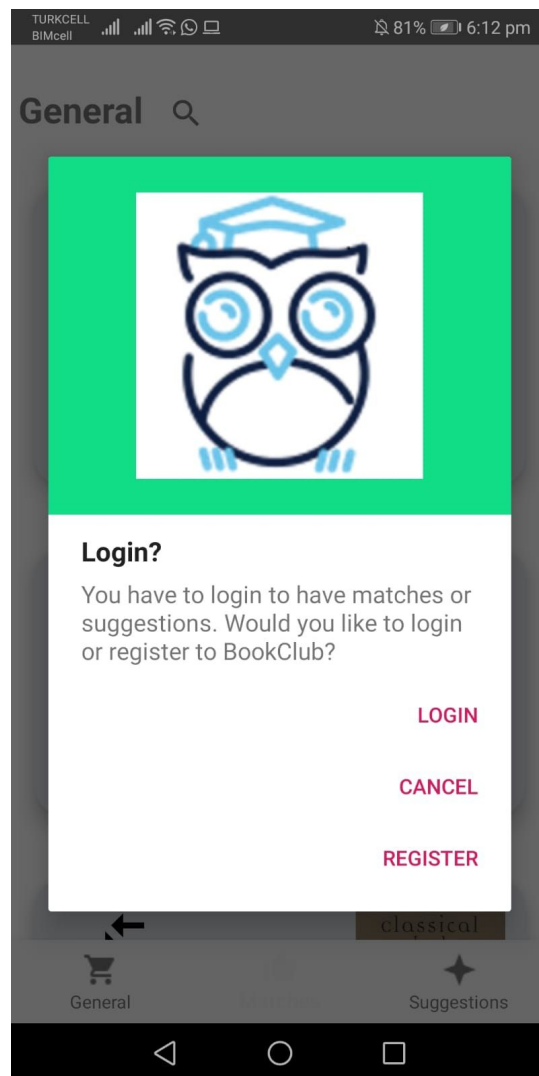
Email

CHANGE PASSWORD

Forgot Password Screen

In this screen, the user can enter either their username or email address which they have already registered with. If their credential is correct, BookClub will automatically send them a password reset email to their mail address which they have registered with. The user can reset their password from that email. Later, they will enter from the login screen again.

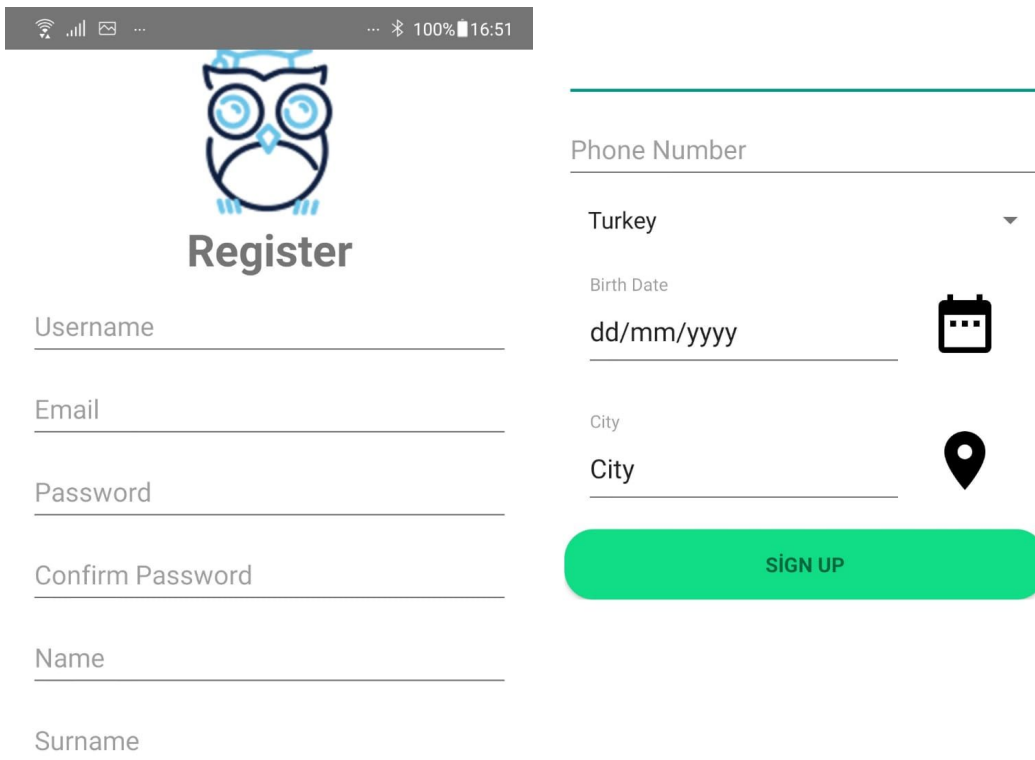
8.4 Login or Register Proposal Page for Guests



Login or Register Proposal Screen

The users who are not registered to the app, but want to see what it is about can enter the application as guests. The guests are limited with the actions, for example they cannot have any matches or suggestions and they cannot add any book to their tradelist or wishlist simply because they do not own those lists. The guest users can only search for the books. If they want to perform some other actions, the system will automatically suggest them to either login or register to the application.

8.5 Register Page



The image shows a mobile application register screen. At the top is a status bar with icons for signal, Wi-Fi, battery, and time (16:51). Below the status bar is a blue owl logo with the word "Register" in bold black text. To the left of the owl are five input fields: Username, Email, Password, Confirm Password, Name, and Surname. To the right of the owl are three input fields: Phone Number, Birth Date (with a calendar icon), and City (with a location pin icon). A green "SIGN UP" button is at the bottom right.

Register

Username

Email

Password

Confirm Password

Name

Surname

Phone Number

Turkey

Birth Date

dd/mm/yyyy

City

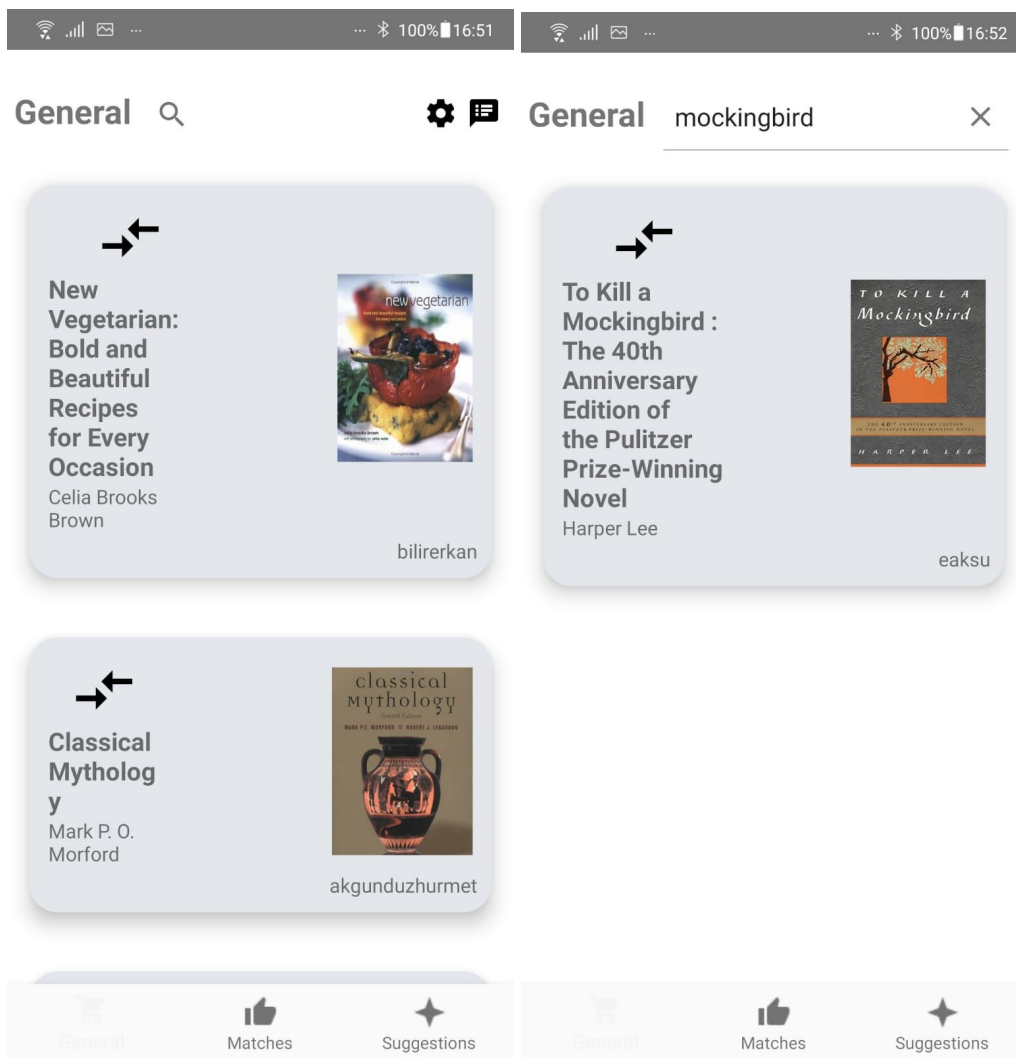
City

SIGN UP

Register Screen

The user is lead to the Register Page after pressing the sign up button in the Initial Main Page. Here the user should enter their personal information and click the sign up button. After successfully signing up, they will be lead to the General Page.

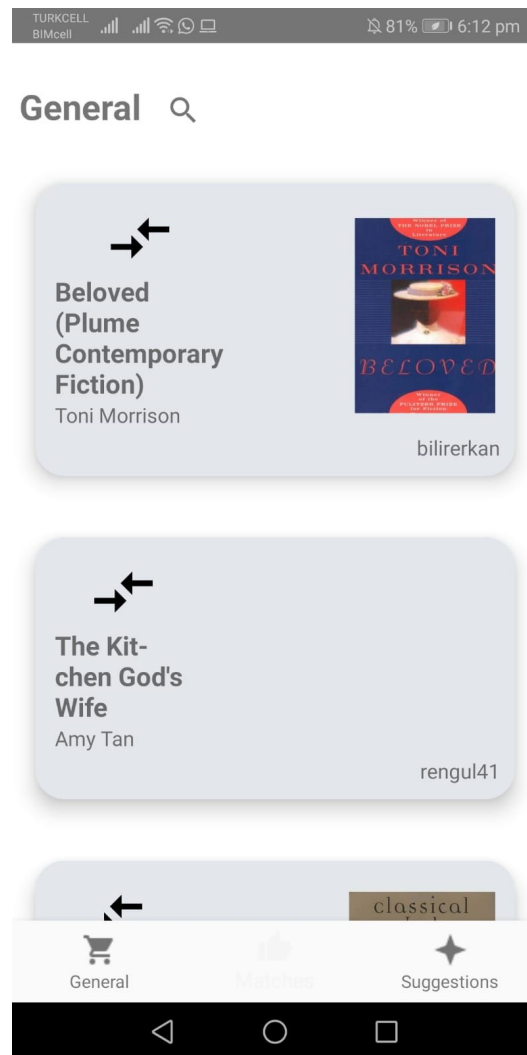
8.6 General Page For Users



General Page Screen for Users

After successfully logging in or signing up, the users are lead to the General Page where they see all the offered books from other users and can also search for a specific book. Additionally, this page is the user's guide, which means they can access everything from here. The user can press to see their chats, settings, matches and suggestions. Each of these buttons will lead the users to the separate pages.

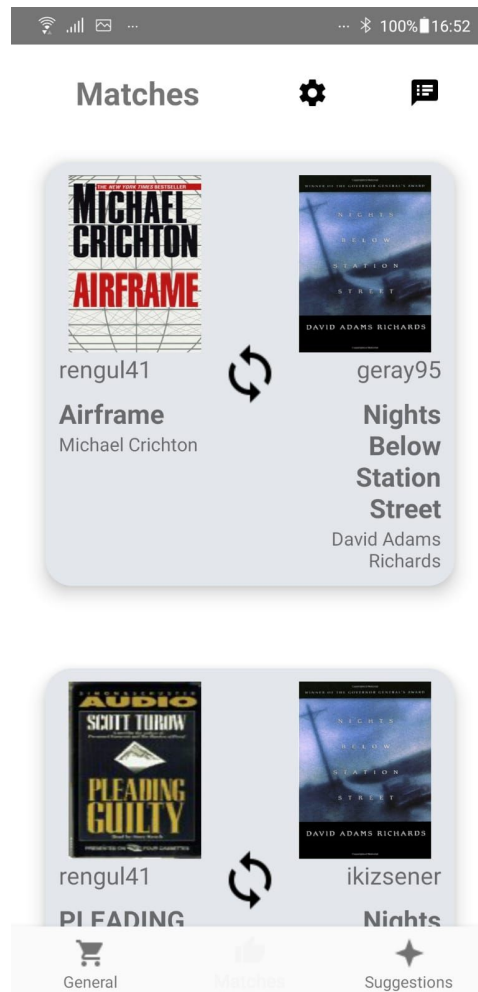
8.7 General Page for Guests



General Page Screen for Guests

If the user enters as a guest, they encounter the page named General Page Screen for Guests. Since its a guest user, they are restricted with the actions. Thus, in this screen the only action the user can do is to search for some books and see the marketplace of books. If they try to perform some other action, BookClub will redirect them to the login page.

8.8 Matches Page



Matches Screen

If the user presses the matches button, they are lead to Matches Page. Here the user sees their top best matches in detail. They see which book should be given and which book will be taken. They also see the user they are matched with. If they press the middle button on the match, they confirm it. If they swipe the match, they reject it. If the other user has already confirmed the match, the chat stream will be opened. Additionally, the user can go to their settings and already existing chats or return to the general page. The user can also go to the suggestions page.

8.9 Suggestions Page

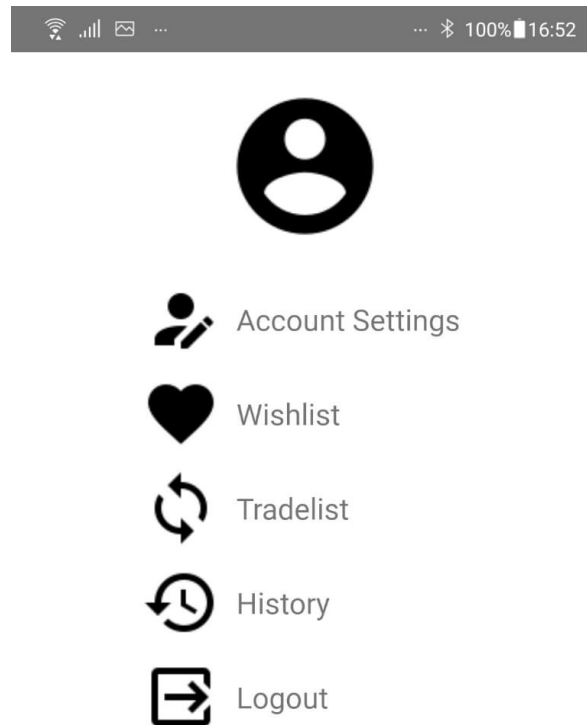


Suggestions Screen

If the user presses the suggestion button, they are lead to Suggestions Page. Here the user sees their top best suggestions in detail. They see which book should be given and which book will be taken. They also see the user they are suggested with. If they press the middle button on the suggestion, they confirm it. If they swipe the suggestion, they reject it. If the other user has already confirmed the suggestion, the chat stream will be opened. Additionally, the user can go to their settings and already existing chats or return to the general page. The user can also go to the matches page.

Suggestions and Matches Pages are very similar, however the algorithm behind them is completely different.

8.10 Settings Page



Settings Screen

If the user presses the settings button in the top of each screen, they are lead to the Settings Page. In this page the user can control their own settings and lists. They can go to the account settings, wishlist, tradelist, history or logout from the application. If the user clicks the picture button, they will be lead to the page where they can view their own profile information. They can also return to the general page.

8.11 Account Settings Page

Account Settings

Name
Abdiş Sezgin

Current Password

New Password

Retype New Password

Birth Date
25/04/1955 **CHANGE BIRTHDAY**

New Email
sakaryasabettin@corlu.com

New Phone Number
+90 (837) 4060352

☒ Availability ☒ Messagable ☒ Last Seen

City
Ahi Evran Osb, Turkey

Account Settings

Current Password

New Password

Retype New Password

Birth Date
25/04/1955 **CHANGE BIRTHDAY**

New Email
sakaryasabettin@corlu.com

New Phone Number
+90 (837) 4060352

☒ Availability ☒ Messagable ☒ Last Seen

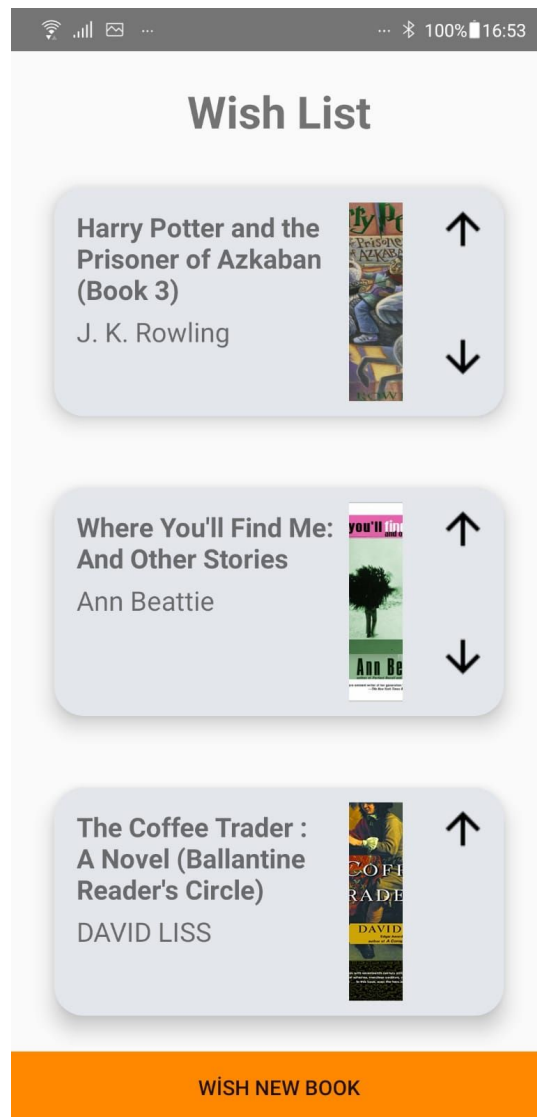
City
Ahi Evran Osb, Turkey

Account Settings Screen

In Account Settings Page, the user can change/update their name, password, mail address, birth date, phone number, availability status, messagability status, last seen status and location.

The user can also return back to the Settings Page.

8.12 Wishlist Page



Wishlist Screen

If the user presses the wishlist button in the Settings Page, they will be lead to the Wishlist Page where they will see their desired books. Here the user can change the books' orders by pressing the arrows next to them. The order of the books highly changes the suggestion and match algorithm for the users. For example, if the user wants the "Harry Potter" book first, the suggested books will probably be very similar to this genre.

The user can add a new book to wishlist from this screen by clicking the “wish new book” button.

8.13 Tradelist Page

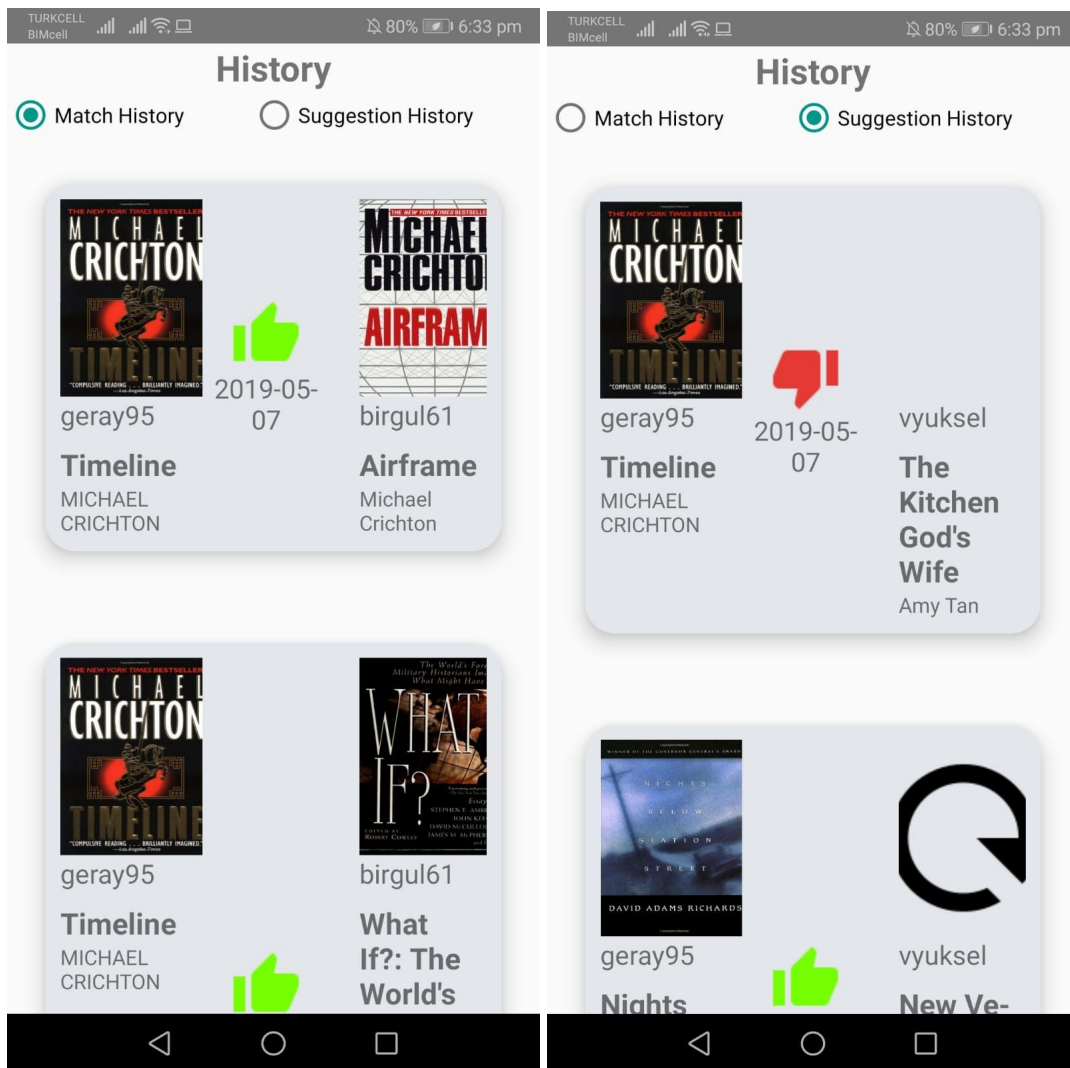


Tradelist Screen

If the user presses the tradelist button in the Settings Page, they will be lead to the Tradelist Page where they will see their giving books.

The user can add a new book to tradelist from this screen by clicking the “trade new book” button.

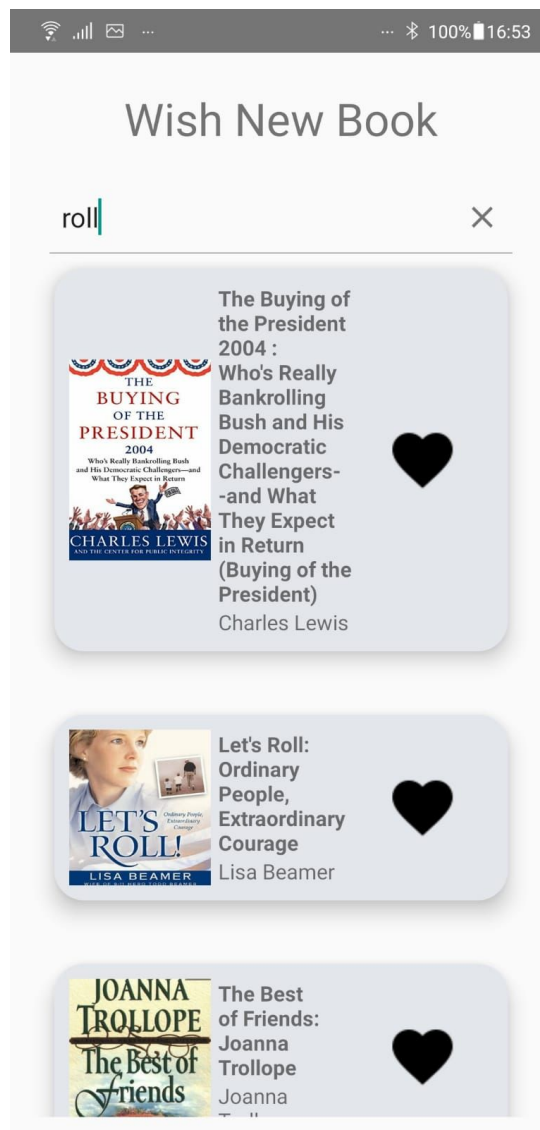
8.14 History Page



User History Screen

If the user presses the history button in the Settings Page, they will be lead to the History Page. Here the user will see all the acitivity they had in the application, such as their reject or confirmed matches/suggestions. They can clear the history whenever they want. By switching the checkboxes, they will either see match activity or suggestion activity.

8.15 Add New Book to Wishlist Page

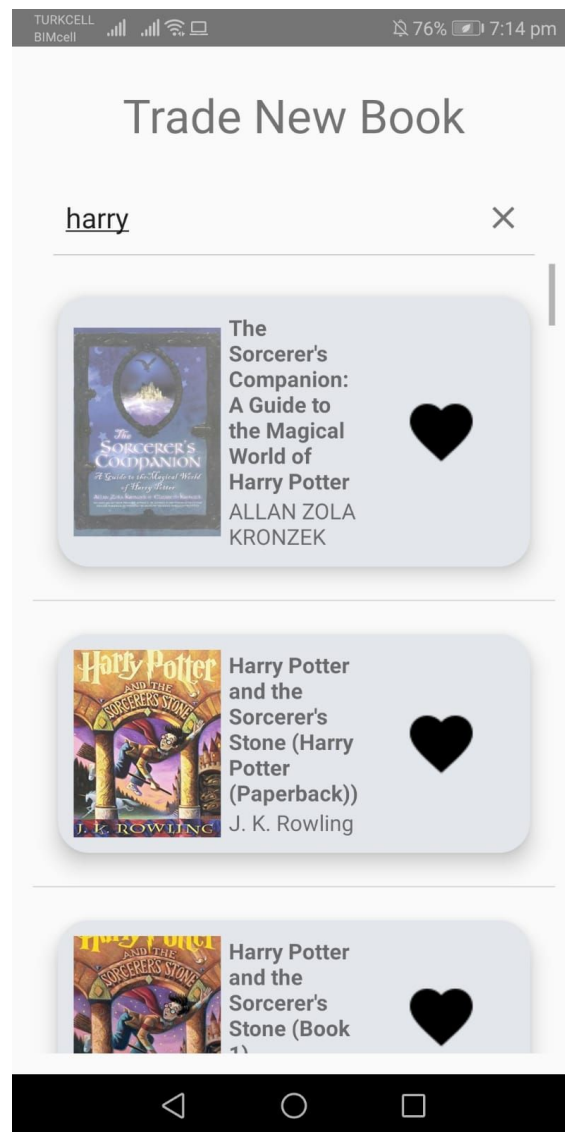


Wish New Book Screen

If the user presses the “wish new book” button in the Wishlist Page, they are lead to the “Add New Book to Wishlist” Page. Here the user searches for the book they would like to get.

The user can also return back to the Settings Page.

8.16 Add New Book to Tradelist Page



Trade New Book Screen

If the user presses the “trade new book” button in the Tradelist Page, they are lead to the “Add New Book to Tradelist” Page. Here the user searches for the book they would like to give. This way, BookClub ensures that the user enters correct information for the book. Later if the user is caught trading not original book, the rating for that user drops and the user may get banned. That’s why BookClub encourages the users to add a book if and only if it is original.

8.17 My Chats Page

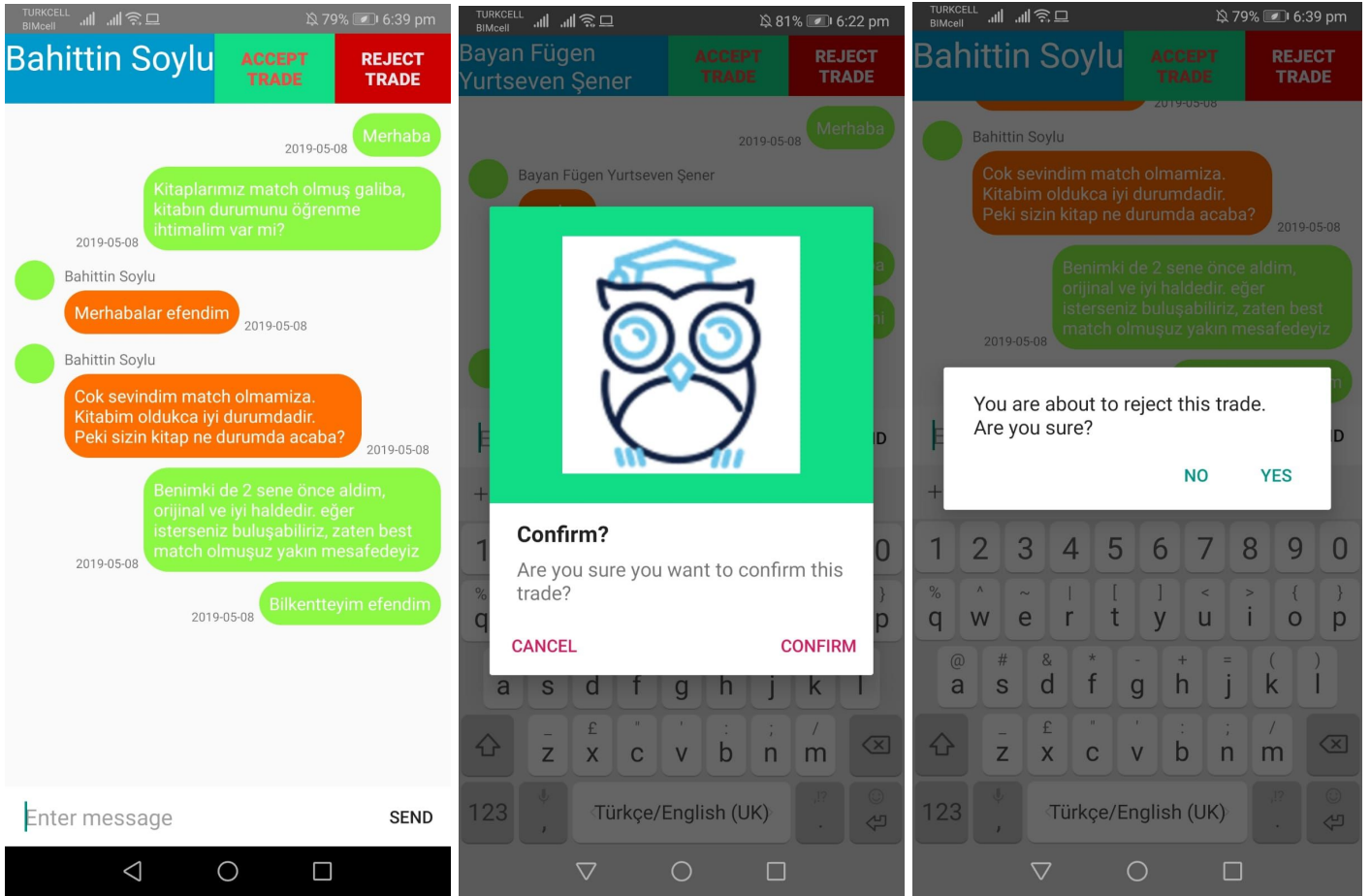


My Chats Screen

If the user clicks the chat button on any page, they are lead to My Chats Page where they can see all chats they are currently having. If a chat does not have a star sign next to it, it means it is not confirmed yet. If there is a star, it means the user should rate the other user, however it is not necessary. To rate the other user, the star should be clicked. By clicking the chat, the user can enter to the messages screen.

If the user clicks the picture of the other user on the chat instance, they will be lead to the page for viewing the other user's profile information.

8.18 Single Chat's Page

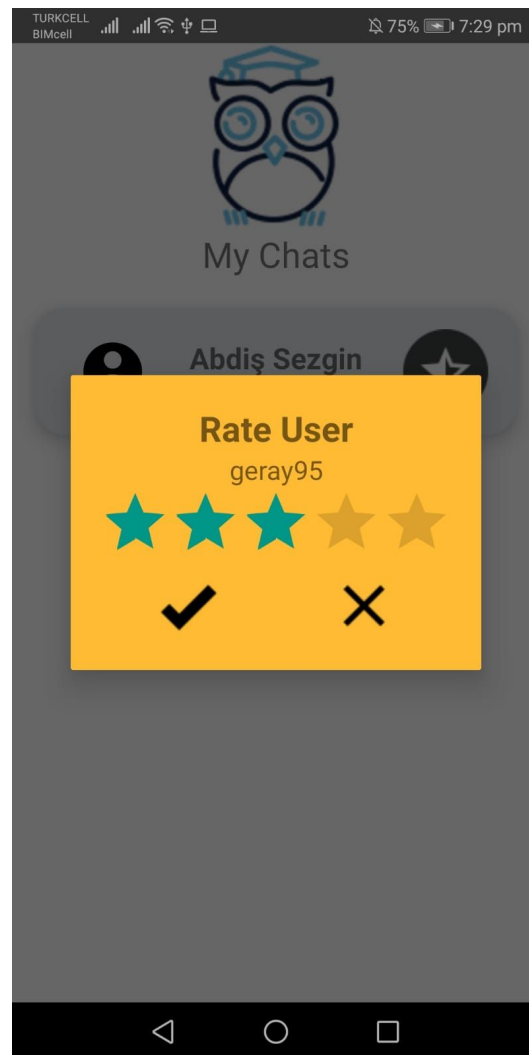


Chat Screen

After clicking the chat instance, the user is lead to the Single Chat's Page where they see the messages between them and the user they are currently trading with.

Additionally, there are two buttons on the top of the messages, which are accepting and rejecting the trade. If the users succesfully traded the books in real life, they need to accept the trade in order update the algorithm. If the trade did not happen and the users are not willing to trade they can reject it. If the both users confirm the trade, the star sign appears and the users proceed to rate each other.

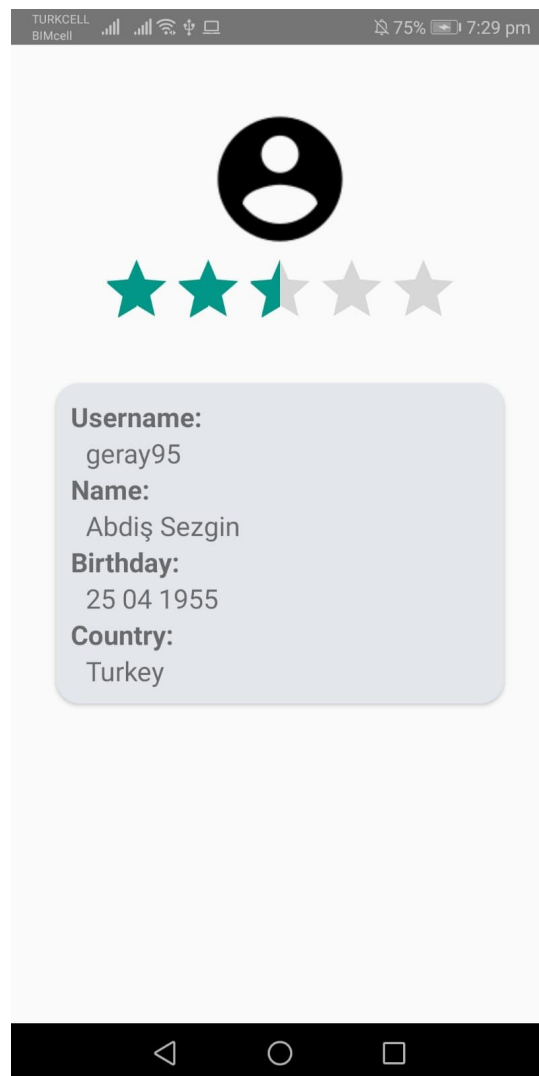
8.19 Rating a User Page



Rating a User Screen

After confirming the trade, the user gets a pop up placeholder for rating the other user. The users can be rated out of five stars. After clicking the check mark, the other user is being rated.

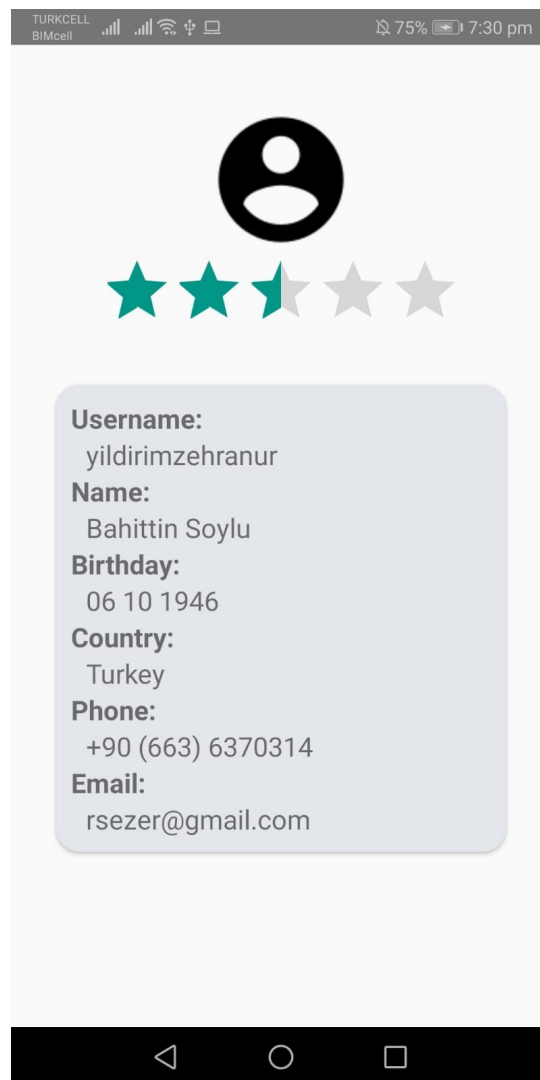
8.20 Other User's Profile Page



Other User's Profile Screen

After pressing the picture button on My Chats screen, the user is lead to the page where they view the other user's profile. Here the user can view the other trader's overall rating, username, name, birth date and country.

8.21 Own Profile Page



Own Profile Screen

After pressing the picture button on Settings screen, the user is lead to the page where they view their own profile information. Here the user can view the information such as username, name, birthday, country, phone and email.

References

- [1] P. Todd Fredrich, "REST API Tutorial", Restapitutorial.com, 2019. [Online]. Available: <https://www.restapitutorial.com/>. [Accessed: 04- May- 2019].
- [2] "The Web framework for perfectionists with deadlines | Django", Django project.com, 2019. [Online]. Available: <https://www.djangoproject.com/>. [Accessed: 02- May- 2019].
- [3] "Postman | API Development Environment", Postman, 2019. [Online]. Available: <https://www.getpostman.com/>. [Accessed: 03- May- 2019].
- [4] "Download Android Studio and SDK tools", Android Developers, 2019. [Online]. Available: <https://developer.android.com/studio>. [Accessed: 09- May- 2019].
- [5] "OkHttp", Square.github.io, 2019. [Online]. Available: <http://square.github.io/okhttp/>. [Accessed: 04- May- 2019].
- [6] "New & Used Books from ThriftBooks | Buy Cheap Books Online". [Online]. Available: <https://www.thriftbooks.com/>. [Accessed: 13 Oct, 2018].
- [7] "BookFinder.com: New& Used Books, Rare Books, Textbooks, Out of Print Books". [Online]. Available: <https://www.bookfinder.com/>. [Accessed: 13 Oct, 2018].
- [8] "API Testing using Postman", Medium, 2019. [Online]. Available: <https://medium.com/aubergine-solutions/api-testing-using-postman-323670c89f6d>. [Accessed: 03- May- 2019].
- [9] "How to Install XAMPP and WordPress Locally on PC/Windows", WPMU DEV Blog, 2019. [Online]. Available: <https://premium.wpmudev.org/blog/setting-up-xampp/>. [Accessed: 03- May- 2019].
- [10] A. Becker, "HeidiSQL - MariaDB, MySQL, MSSQL and PostgreSQL made easy", HeidiSQL.com, 2019. [Online]. Available: <https://www.heidisql.com/>. [Accessed: 09- May- 2019].
- [11] "Genymotion – Fast & Easy Android Emulator", Genymotion – Android Emulator for app testing, 2019. [Online]. Available: <https://www.genymotion.com/>. [Accessed: 05- May- 2019].

- [12] "Picasso", Square.github.io, 2019. [Online]. Available: <https://square.github.io/picasso/>. [Accessed: 04- May- 2019].
- [13] "hudomju/android-swipe-to-dismiss-undo", GitHub, 2019. [Online]. Available: <https://github.com/hudomju/android-swipe-to-dismiss-undo>. [Accessed: 03- May- 2019].
- [14] "d-max/spots-dialog", GitHub, 2019. [Online]. Available: <https://github.com/d-max/spots-dialog>. [Accessed: 05- May- 2019].
- [15] "yarolegovich/LovelyDialog", GitHub, 2019. [Online]. Available: <https://github.com/yarolegovich/LovelyDialog>. [Accessed: 03- May- 2019].
- [16] "Python Data Analysis Library — pandas: Python Data Analysis Library", Pandas.pydata.org, 2019. [Online]. Available: <https://pandas.pydata.org/>. [Accessed: 05- May- 2019].
- [17] "factory_boy — Factory Boy latest documentation", Factoryboy.readthedocs.io, 2019. [Online]. Available: <https://factoryboy.readthedocs.io/en/latest/>. [Accessed: 03- May- 2019].
- [18] "The Web framework for perfectionists with deadlines | Django", Django project.com, 2019. [Online]. Available: <https://www.djangoproject.com/>. [Accessed: 05- May- 2019].
- [19] "django-cors-headers", PyPI, 2019. [Online]. Available: <https://pypi.org/project/django-cors-headers/>. [Accessed: 05- May- 2019].
- [20] T. Christie, "Home - Django REST framework", Django-rest-framework.org, 2019. [Online]. Available: <https://www.django-rest-framework.org/>. [Accessed: 05- May- 2019].
- [21] J. Biggs and J. Biggs, "There Is One New Book On Amazon Every Five Minutes – TechCrunch," *TechCrunch*, 22-Aug-2014. [Online]. Available: <https://techcrunch.com/2014/08/21/there-is-one-new-book-on-amazon-every-five-minutes/>. [Accessed: 09-May-2019].
- [22] D. Arias, "Hashing Passwords: One-Way Road to Security", *Auth0 - Blog*, 2019. [Online]. Available:

<https://auth0.com/blog/hashing-passwords-one-way-road-to-security/>. [Accessed: 03-May- 2019].

[23] D. Arias, "Adding Salt to Hashing: A Better Way to Store Passwords", *Auth0 - Blog*, 2019. [Online]. Available: <https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords/>. [Accessed: 03- May- 2019].