

UKRAINIAN CATHOLIC UNIVERSITY

BACHELOR THESIS

Mobile application for UCU Library

Author:
Kateryna DETSYK

Supervisor:
Serhii MISKIV

*A thesis submitted in fulfillment of the requirements
for the degree of Bachelor of Science*

in the

Department of Computer Sciences
Faculty of Applied Sciences



APPLIED
SCIENCES
FACULTY ●

Lviv 2022

Declaration of Authorship

I, Kateryna DETSYK, declare that this thesis titled, "Mobile application for UCU Library" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

"Books are a uniquely portable magic."

Stephen King

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Bachelor of Science

Mobile application for UCU Library

by Kateryna DETSYK

Abstract

Not so long ago, calculators, photo galleries, books, photo cameras, and even a video camera were physically different, unrelated objects. However, with the advent of smartphones, they have all become available in small, compact devices that you can always carry with you wherever you go. Smartphones offer access to many different features. I think the most important of them is access to the information.

It is highly important for students and all people of the 21st century to have quick access to information. Not all books are freely available on the Internet, and it is not always possible to go to the library to bring a paper book or take a large number of books with you. These are just a few issues that the electronic library on your smartphone can solve.

Registered users of the library of Ukrainian Catholic University will be able to read books on their smartphones, no matter when and no matter where they are if they have access to the Internet.

Acknowledgements

First of all, I would like to thank my supervisor Serhii Miskiv for the support and good advice. I am also very grateful for the help in finding the topic. Because finding a topic for the thesis was a pretty difficult thing for me.

My sincere thanks to the UCU University and the UCU Library for the opportunity to work on this project. I like books, both paper and electronic. Reading is an integral part of my life, so I am very happy to have worked on this project. Big thanks to Andriy Stankevych, Olexandra Yaruchyuk, Oksana Mykytyn, Oleh Lahodniuk, Volodymyr Bokla and Roman Pototsky.

Contents

Declaration of Authorship	i
Abstract	iii
Acknowledgements	iv
1 Introduction	1
1.1 Overview	1
1.2 Motivation	1
1.3 Goals	2
1.4 Structure	3
2 Background	4
2.1 Library resources	4
2.1.1 Koha	4
2.1.2 Koha REST API	4
2.1.3 Dataset	5
2.1.4 Server	5
2.2 Flutter	7
2.2.1 Pros of Flutter	7
2.2.2 Cons of Flutter	7
2.3 Python Flask	8
3 Related works	9
3.1 First Example	9
3.2 Second Example	10
3.3 Conclusions on related works	10
4 Application Implementation	11
4.1 User Flow	11
4.2 UI	12
4.3 State Management	13
4.3.1 Flutter BLoC library	14
4.4 Repository	16
4.4.1 MySQL Repository	16
4.4.2 API Repository	17
4.4.3 Service locator	18
4.4.4 Secure Storage	19
4.5 Navigation	19
4.6 Screens	21
4.6.1 Splash Screen	21
4.6.2 Login Screen	22
4.6.3 Home Screen	23

4.6.4	Borrowed Books Screen	25
4.6.5	Book Screen	26
4.6.6	Reading Screen	27
5	API Implementation	28
5.1	Advantages of using API	28
5.2	Connection to the database	29
5.3	Functionality	29
5.3.1	Login	29
5.3.2	Search	30
5.3.3	Recent	31
5.3.4	Borrowed	31
6	Conclusion	33
6.1	Result summary	33
6.2	Future works	33
	Bibliography	34

List of Figures

2.1	Opening connection to the server	6
2.2	SSH Tunneling tab	6
2.3	Shared Shell	7
3.1	Digital Library screens from Play Market	9
3.2	Koha OPAC screens from Play Market	10
4.1	User Flow	12
4.2	UI Design	13
4.3	BLoC Architecture	14
4.4	BLoC Architecture Workflow	16
4.5	Workflow with MySQL Repository	16
4.6	Workflow with API Repository	17
4.7	Navigation	20
4.8	Splash Screen	22
4.9	Login Screen	23
4.10	Login Screen show password	23
4.11	Login Screen alert popup	23
4.12	Login Screen validation	23
4.13	Home Screen	24
4.14	Home Screen, search results	24
4.15	Home Screen, no results	24
4.16	Home Screen, loading	24
4.17	Borrowed Books screen	25
4.18	Borrowed Books screen, expired books	25
4.19	Books screen, reading available	26
4.20	Book screen, reading not available	26
4.21	Reading screen	27
4.22	Reading screen, choose page	27
5.1	Communication between mobile application and database via API	28

List of Tables

4.1	Navigator push functions' stacks comparison	21
5.1	API login responses	30
5.2	API search responses	31
5.3	API recent responses	31
5.4	API borrowed responses	32

List of Abbreviations

UCU	U krainian C atholic U niversity
ILS	I ntegrated L ibrary S ystem
PDF	P ortable D ocument F ormat
API	A pplication P rogramming I nterface
REST	R epresentational S tate T ransfer
SQL	S tructured Q uery L anguage
id	i dentify d ocument
JSON	J ava S cript O bject N otation
WSGI	W eb S erver G ateway I nterface
SSH	S ecure S hell
BLOB	B inary L arge O bject
BLoC	B usiness L ogic C omponent
UI	U ser I nterface
URL	U niform R esource L ocator
APK	A ndroid P ac K age
HTTP	H yper T ext T ransfer P rotocol
HTTPS	H yper T ext T ransfer P rotocol S ecure

Dedicated to my Family, those who always support me and believe in me, even when I don't believe in myself.

Chapter 1

Introduction

This chapter provides an overview of the Ukrainian Catholic University(UCU) Library. It describes the problem of publishing e-books that the UCU library has faced. It tells the reasons why electronic libraries are relevant in our times. The chapter describes the motivation for creating a mobile application to overcome these problems and defines the main goals of the thesis.

1.1 Overview

The library of Ukrainian Catholic University is located in the Sheptytsky Center and in the building of the Faculty of Philosophy and Theology. The library offers its users a lot of different books. Currently, the information site of the library works as a part of the site of the Sheptytsky Center¹. Here library visitors can learn about library events, opening hours, rules of use and other information. Last year, UCU students developed a new separate information site for the UCU library with a comfortable design and user-friendly interface. However, the site is not finished yet, and it is still not freely available. If the users wants to find out whether there is a desired book in the library or check on which bookshelf it is located, they must use the electronic catalog Koha UCU². It is also possible for authorized users to make a booking. However, none of these web resources provides an opportunity to read books online.

1.2 Motivation

The UCU Library has long planned to switch to digital format, but they did not have a suitable platform to offer e-books. In recent years, the need for electronic libraries is growing as well as the need for remote access to educational materials and information in general. The deadly virus, quarantine, distance learning, and war, forced us to face enormous challenges. It is not always possible to get to the paper book lying on the shelf in the Sheptytsky Center. The Internet considerably simplifies access to information. A huge bookstore in a small gadget can save you time searching for a physical book. There are many reasons to create an electronic library on a smartphone. There are many problems that such a compact library can solve :

- Not all users of the UCU library, including students, live in Lviv. They can live in other cities or even other countries. In the conditions of distance learning, quarantine or vacation, they may not have the opportunity to come to the library to pick up the book they need to study. E-library can make books from UCU Library available for them.

¹<https://center.ucu.edu.ua/biblioteka/>

²<https://opac.ucu.edu.ua/>

- The number of copies in the library is limited. The capacity of the library is limited by the number of bookshelves in the Sheptytsky Center. The library purchases fewer copies of one book but more different books to increase the range of books on the shelves. If UCU Library is available in electronic format, the students preparing for an important exam will be able to read a book in PDF format simply from their smartphones. And they should not worry that there will not be a copy for them in the library.
- Another problem is that some of the books are stored in the building of the Faculty of Philosophy and Theology. And UCU library has no book delivery from one building to another.
- One more important aspect is spending time searching for a book and getting to the library, for example, from the other side of the city. Students and other holders of a reader's tickets will be able to read books at any time and anywhere if they have access to the Internet. The time they saved, they could spend more productively, for instance, doing their homework.

Of course, there are many more problems that an electronic library in a smartphone can solve, here are just a few of them.

Another significant reason to create an e-library is the relevance of mobile applications. Most people carry smartphones with them everywhere. They are compact and very functional. You can open the desired book and read it just in a cafe sitting at a table, or when you travel. If the application is relevant, comfortable, and does not take up much space on the device, it has high chances of being liked by users. Furthermore, most UCU library users visit the Koha UCU catalog from their smartphones.

1.3 Goals

My thesis aims to create a mobile application for UCU Library readers. All users will have the opportunity to search for books and browse the catalog. Registered holders of reader's tickets will be able to read all available e-books and see a list of books they borrowed.

Functionality to be implemented in the application in the framework of the thesis:

- **Login** - holders of reader's tickets can enter the login and password to access e-books.
- **Book Search** - anyone, not just logged-in users, will be able to search for books.
- **Borrowed Books** - authenticated users will see the list of borrowed books.
- **Book Information** - all interested users will be able to view information about the book, title, author, the number of pages, and others.
- **Book Reader** - logged-in users can read the book in PDF format, provided that the PDF file of this book is available in the database.

1.4 Structure

- **Chapter 1 Introduction** - describes the problem statement and the motivation to create a mobile application to solve this problem. And it defines the goals of the thesis.
- **Chapter 2 Background** - describes the technologies, services, data, and other stuff that were used for the implementation of a mobile application.
- **Chapter 3 Relative works** - This section describes two applications most similar to my application. It also highlights their pros and cons and why my application has an advantage over them.
- **Chapter 4 Application Implementation** - describes the implementation of this application in detail.
- **Chapter 5 API Implementation** - performs the overview of the new API for this app and Koha database and describes the implementation.
- **Chapter 6 Conclusions** - summarizes the results and describes the future works.

Chapter 2

Background

The second chapter provides an overview of library resources necessary for this application development. It also describes the technologies chosen to implement this mobile application and API.

2.1 Library resources

This mobile application has to work with library resources such as a book catalog or a data set with library users. During the development, I got acquainted with the UCU Library system ILS(Integrated Library System) Koha¹ in more detail, as my work intersected with it. The application works with the same database used by the electronic catalog Koha UCU.

2.1.1 Koha

ILS Koha is an automated web-based open source library system for accounting books and library users. The name "koha" means "gift" in the Maori language. It provides opportunities to search, catalog, and track any number of readers, book copies, and other data. Registered library users can keep a reading list and rate books. And librarians can enter all the necessary information about books and users in the database.

There are some pros and cons of using ILS Koha. Some disadvantages complicate the use of Koha or even complicate the opportunity of expanding its functionality. ILS Koha was built with the Perl programming language back in 1999. Implementation in Perl makes it difficult to change or update the code because not many programmers use this language. Thus, it is almost impossible to expand the backend of the catalog to add a function to read books online.

Therefore, the mobile application has the potential to become popular among users as a more user-friendly and functional environment.

2.1.2 Koha REST API

ILS Koha has its own REST API(Representational State Transfer Application Programming Interface)² also implemented in Perl. API offers some functionality, for instance getting information of book location : `/api/v1/items/{item_id}` or information about library patrons : `/api/v1/patrons/{patron_id}`.

However, this API does not provide some useful functions, including those necessary for the mobile application. There is no method for authentication, so it is

¹<https://koha-community.org/>

²<http://koha.ucu.edu.ua/api/v1/.html#op-get-holds>

impossible to develop a login using this API. There is also no method to get information about books such as author, title, pages, and others. Such a necessary search method is also not implemented there.

Unfortunately, it is impossible to update this API because its implementation in Perl is very complicated.

I created a new API to get access to the Koha database. In chapter 5, you can read about the new API in more detail.

2.1.3 Dataset

API fetches necessary data from the MySQL database of ILS Koha. Our mobile application will need information about books and library users from the database.

To provide authentication, we need to know users' ids and passwords. Since email is unique for everyone, it is used as an id (Identity document) in the database. User passwords are stored in the hashed form to make a system more secure.

Book information is divided into several tables in the database. We need three of them. Basically, we need tables that contain information about the book's title, the author, the image, the place of storage, and a link to the PDF file of the book.

2.1.4 Server

ILS Koha UCU, with its database, is located on the virtual server that uses Linux Debian 9 operating system.

But before using actual data from the production server, I use a testing server with a copy of the library system during application development. I was given all necessary data for connection to the virtual server via SSH (Secured Shell).

SSH - Secure Shell is a protocol that allows securely connecting to the remote computer or server and manipulating server typing commands on your local computer. SSH server that works on the virtual server. It listens to the clients' connections and after successful authentication starts serving the client. SSH client is used to connect to the SSH server and execute commands. For connection I needed a host, port, user and password. To open a connection from the computer with Windows 10 operating system, I use the open-source application PuTTY³.

The connection to the server was needed to access the database on that server. It was impossible to access the database directly from the code because the 3306 port was closed due to security reasons. I used the SSH Tunneling method to access the database and redirect the port.

Firstly, I need to enter the host and port in the corresponding input fields. See the figure 2.1.

Then I need the Tunneling tab. It can be found : Connection/SSH/Auth/Tunnels. Then, I needed to specify the source port and destination, the same as in the figure 2.2. Then press the Add button and the Open button.

Then in the console, I needed to put down the username and password. See figure 2.3. After these steps, I could access the server. That also allowed me to connect to the database directly from the application source code, just using localhost, port 3307, database user, and its password. It was useful when I was creating a new API locally on my computer. However, after the deployment of the new API on the server, the need to use this connection method has disappeared.

³<https://www.chiark.greenend.org.uk/~sgtatham/putty/>

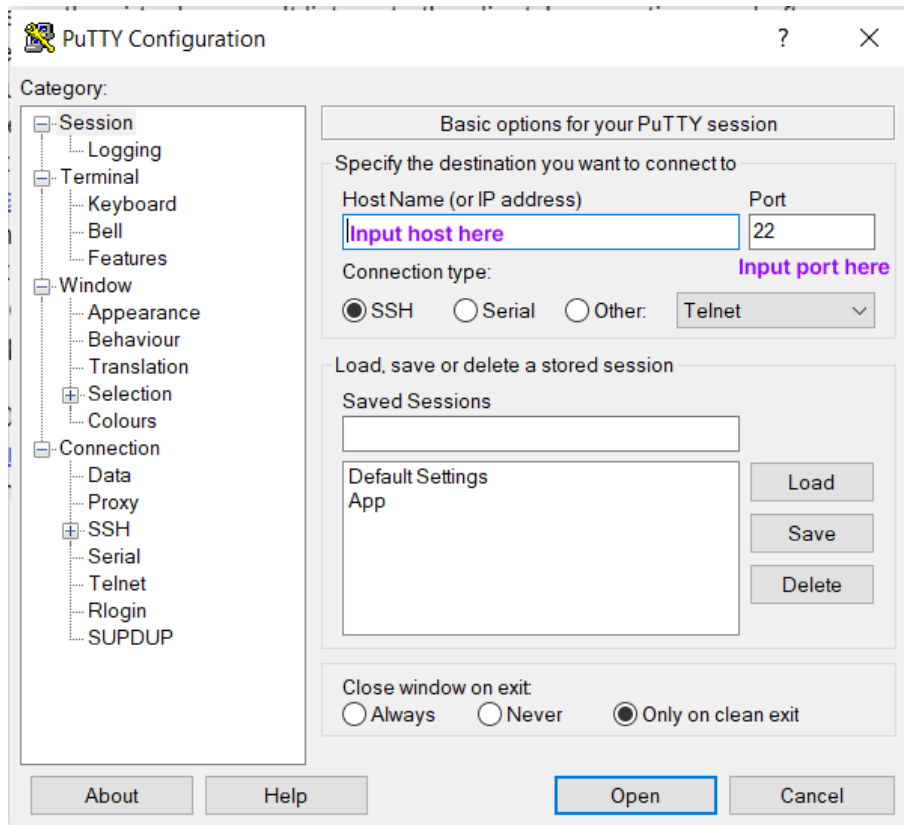


FIGURE 2.1: Opening connection to the server

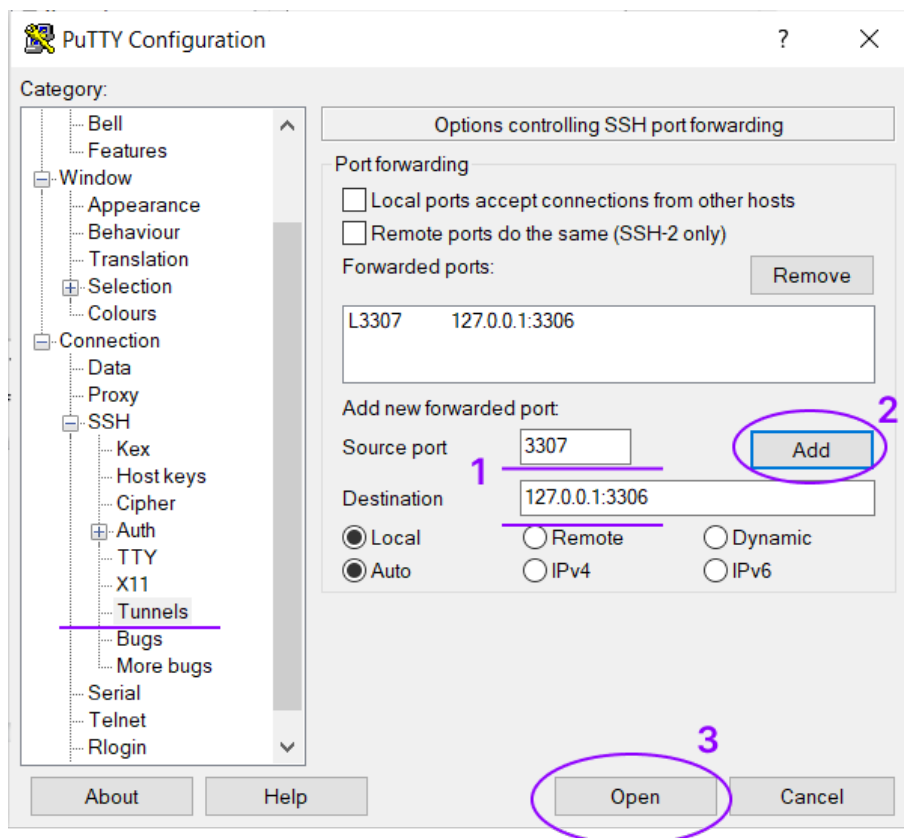
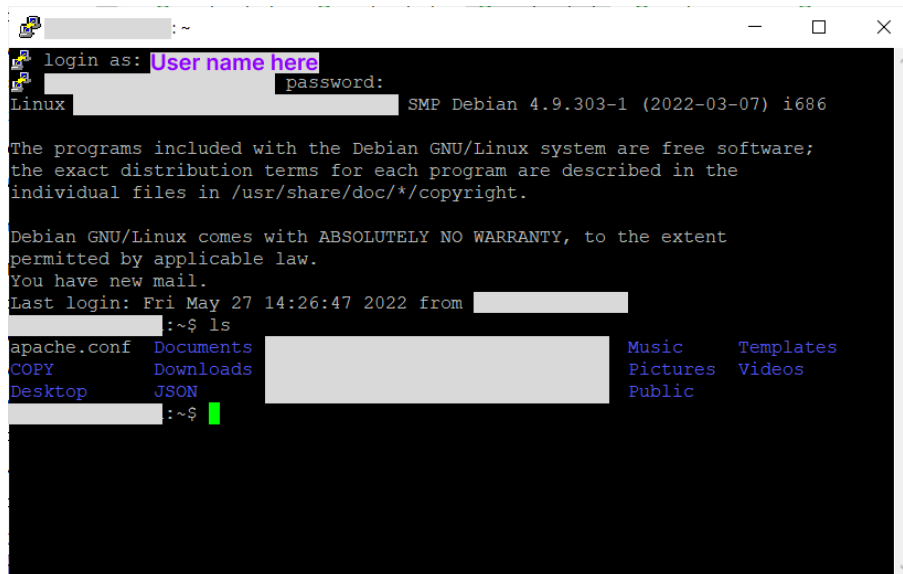


FIGURE 2.2: SSH Tunneling tab



```
login as: User name here
password:
Linux SMP Debian 4.9.303-1 (2022-03-07) i686

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
You have new mail.
Last login: Fri May 27 14:26:47 2022 from [redacted]
[redacted]:~$ ls
apache.conf  Documents  [redacted]  Music      Templates
COPY         Downloads  [redacted]  Pictures   Videos
Desktop     JSON       [redacted]  Public
```

FIGURE 2.3: Shared Shell

2.2 Flutter

Flutter^[8] is an open-source UI framework developed by Google. It was released in 2015 at the Dart developers summit. Flutter is mainly used for building iOS, Android, and web applications. In addition, version 2.2 gave early access to creating desktop applications for Windows, Mac, and Linux. The biggest advantage of Flutter is that a single codebase can be run on different platforms.

The Flutter framework is based on the Dart language. For Android and iOS, it compiles into machine code. For the Web, the Dart translates into JavaScript, and for desktop applications, it uses Dart Virtual Machine.

One of the most useful Flutter features is “hot-reload”. It allows you to rebuild the screen without recompiling the whole project. Any changes in source code will be immediately seen on the running application without losing state and without restarting.

2.2.1 Pros of Flutter

- Flutter is the best technology for multi-platform development.
- It is an open-source technology.
- ‘Hot reload’ feature makes development faster and more comfortable.
- Flutter framework consists of a big library of widgets that can be personalized as developers want.
- There are also a lot of packages and plugins that can be useful for developers.

2.2.2 Cons of Flutter

Although Flutter is a very comfortable technology, there are still some disadvantages. But I think in this case advantages outweigh disadvantages.

- Even if it is a cross-platform technology, developers still need to think about UI adaptation for different screen sizes. Applications for desktop and applications for mobile still have differences in the UI.
- Not all Flutter packages satisfy all platforms. And when developers create a package it should at least satisfy Android and iOS requirements.

2.3 Python Flask

As ILS Koha REST API was not suitable for the application needs and that it was impossible to update it, I had to create a new REST API. It would help my application communicate with the library database. This mobile application would not make any updates to the database. It needs only to take the information. So, I needed to create only to get requests. It is usual for mobile application development if the data comes in JSON (JavaScript Object Notation) format.

For the new API implementation, I chose Flask [7]. Flask is a micro web framework realized in Python. Based on Werkzeug WSGI (Web Server Gateway Interface) toolkit and Jinja templates engine. WSGI is a calling convention between the Python web application and web server, in our case it was Apache.

This time, I was working with Python 3.5.3 version and Flask 1.1.4, which was installed on the virtual server of UCU Library.

Chapter 3

Related works

Market research is an important part of new product development. Ready-made potential competitors can be there. Identifying strengths and weaknesses in those existing solutions can help us create a better product based on market needs.

Looking for products similar to our new application, I decided to overview not just applications for reading books, but applications related to the ILS Koha. Although a lot of libraries all over the world use the Koha library system, I found only two such mobile applications available on the Play Market.

3.1 First Example

The closest example of such an application was Digital Library¹. It allows access to the Pakistan Libraries that use ILS Koha. Unauthenticated users have the opportunity to search books. Logged-in users can make a booking. This app just copied the functionality of web ILS Koha for users : searching books, authentication, managing booking, reservations, and others.

Pros of this application: a few libraries can use it.

Cons: that application does not allow e-book reading and has a quite old-fashioned design.

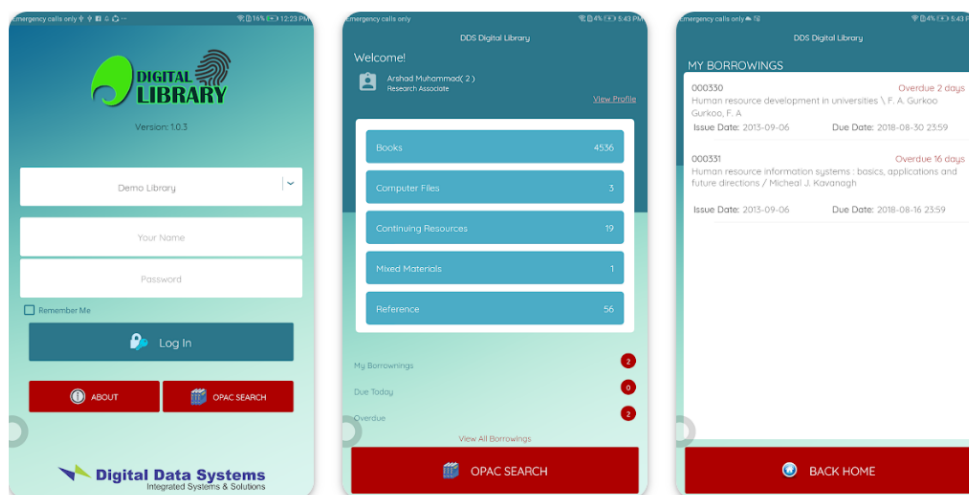


FIGURE 3.1: Digital Library screens from Play Market

¹<https://ddspak.com/dds/koha-mobile-app/>

3.2 Second Example

One more application developed for libraries using ILS Koha is Koha OPAC². This application also has the same functionality as web ILS Koha. You can search for books there, see your list of borrowed books, your reading history, and others. However, this application does not allow reading books online.

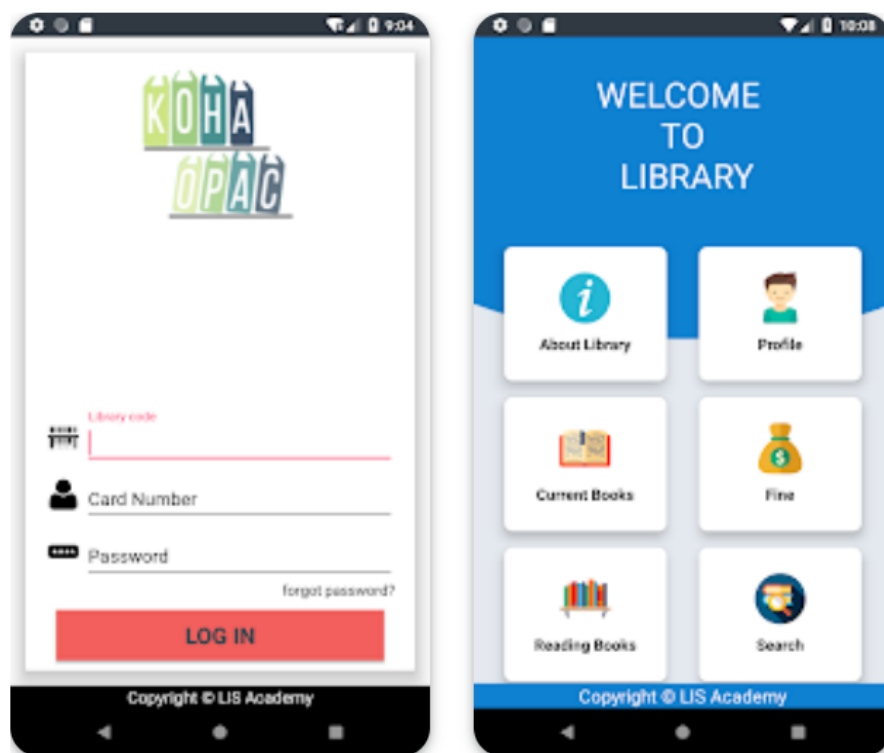


FIGURE 3.2: Koha OPAC screens from [Play Market](#)

3.3 Conclusions on related works

Therefore none of these applications allows you to read books online. They are just mobile versions of ILS Koha and they do not expand its functionality.

Looks like no application works with ILS Koha and provides digital library features. That means that our new mobile application is unique without any analogs.

²<https://play.google.com/store/apps/details?id=com.lisacademy.org.kohaopac&gl=US>

Chapter 4

Application Implementation

This chapter describes how the library's mobile application was created. The functionality of the application and its implementation are considered here. The chapter also provides the application screens' functionality and appearance, implementation of navigation, application architecture (state management), and how the back-end part of the application was handled.

4.1 User Flow

At the beginning of the application implementation path, I had to decide what users and stakeholders wanted to see in the library application. And what exactly should be implemented in this application.

I used the User Flow technique to determine the application's structure and convey the process of working with it. User Flow shows the sequence of actions that users can perform in the application. This technique is often combined with wireframes and not just presented as a chart. But in our case, it is just a chart that spells out the possible steps of our user. This chart uses various geometric shapes to visually convey the beginning of the flow, decision-making moments, errors, transition to screens, and others.

Why is this technique useful?

- Such a scenario scheme will help to agree with stakeholders, potential users, and developers on the structure and functionality of the product.
- After creating a User Flow diagram, you can see where the path is too confusing and where it should be simplified. You can notice the possible mistakes before you start developing an application.
- The diagram shows whether all processes in the application have a logical conclusion and whether they are in principle appropriate.

On our User Flow chart 4.1 we can see :

- To cover all necessary functions we had to implement six different screens. They are filled with violet color on the chart.
- All screens would have a back button to return to the previous screen, except Splash Screen and Home Screen.
- There are branches for authorized users and non-authorized users. Some features such as reading books will be closed to unauthorized users.
- Our chart is not too big and confusing. This means that the structure of the application will be pretty easy to use.

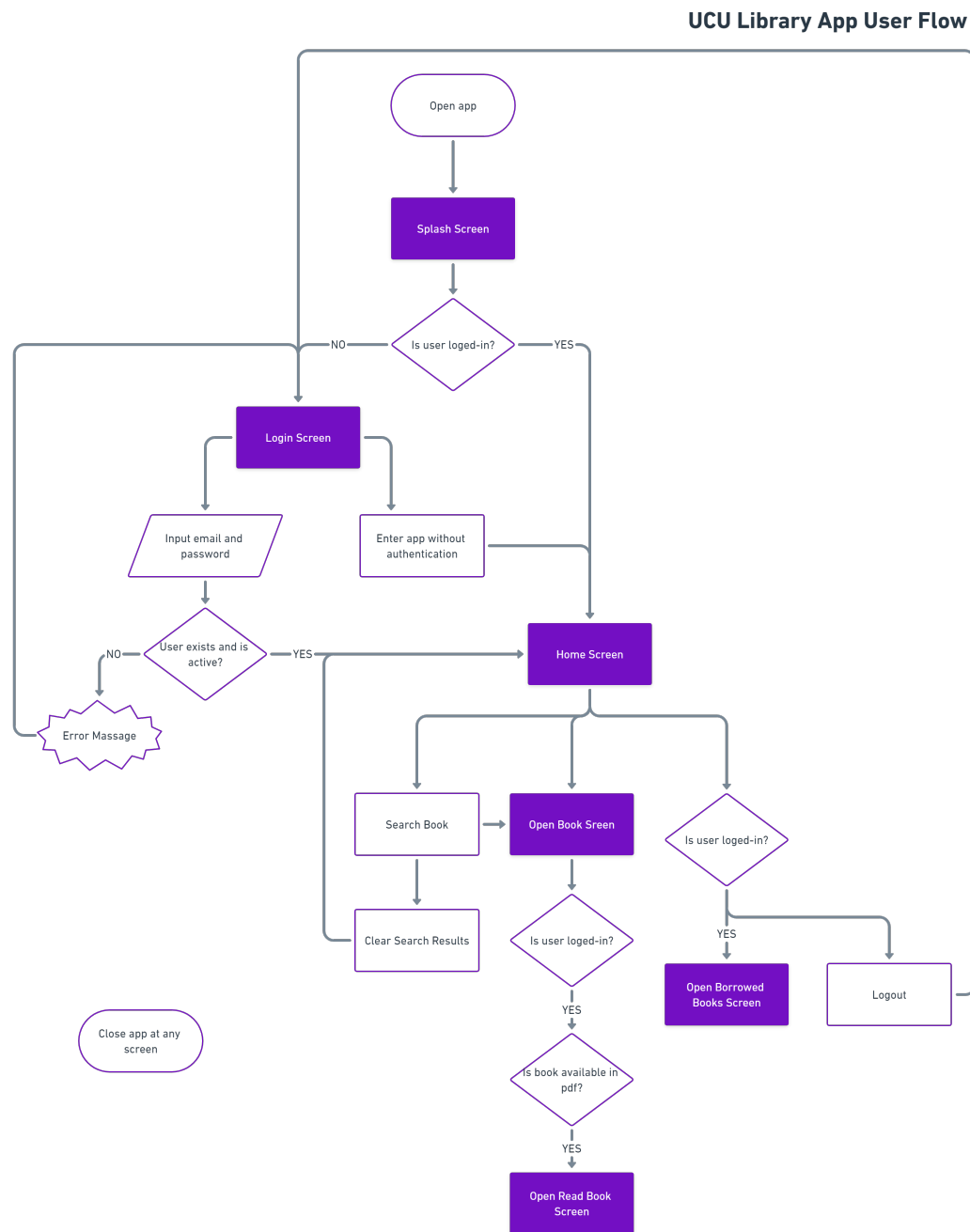


FIGURE 4.1: User Flow

4.2 UI

An essential requirement for the UI (User Interface) design of the application was that it must be in the same style as the new library site. Therefore, to create an application design, I used the same research results that were conducted for the UCU Library website. That design was developed by the students of the Applied Science Faculty in the summer of 2021. I was familiar with the requirements, because I also participated in it. Through surveys, brainstorming, and various studies, we have developed a palette of the site and adapted the design to the mission of the

UCU Library: openness, minimalism, accessibility, and even visual inclusiveness, avoiding outdated library templates that exist in Ukraine.

- The main principles were minimalism and accessibility.
- The color scheme was the same as it was on the library site.
- Except for the font colors, unlike dark gray, I use black to make the text more contrasted.
- On the Borrowed Books Screen, overdue books' cards would be colored light red.
- Font family - Fira Sans.
- To make an accent on some important text, like titles, I use semibold font-weight or medium font-weight.
- Icons were taken from the standard Flutter Icons class¹.
- The application language is Ukrainian.



FIGURE 4.2: UI Design

4.3 State Management

For the development of this application, I chose BLoC state management. BLoC (Business Logic Component)[5] is a state management architecture designed by Google

¹<https://api.flutter.dev/flutter/material/Icons-class.html>

developers and presented in 2018 during Dart Conferention. This architecture is recommended for Flutter application development. The BloC approach aims to separate Business Logic from UI.

This approach is useful because the separation of UI and Business Logic makes updating, testing, and debugging code easier. UI can be easily updated without disturbing Business Logic and vice versa. Developers would be able to know the application state at every moment. Bloc is based on Streams and used for reactive programming. "Reactive programming is programming with asynchronous data streams." [3]

As it was explained in the article "Reactive-Programming-Streams-BLoC" [3], Streams are usually visualized like a pipe. The information comes on the one end of the pipe, then something can happen to this information inside, and then it gets out from the second end of the pipe. Streams can have a few listeners that can be notified when the stream has some new data. Listeners can react to this notification. For instance, widgets can rebuild themselves. A Sink is an abstraction used to send data to the Steam.

In figure 4.3, we can see the visualization of this process. Some widgets can send events to the BLoC via Sink. Then something happens with this data in the BLoC. However, UI does not take part in it. When all operations with data are over, widgets are notified by a BLoC via Steam. As we can see, UI and Business Logic are separated.

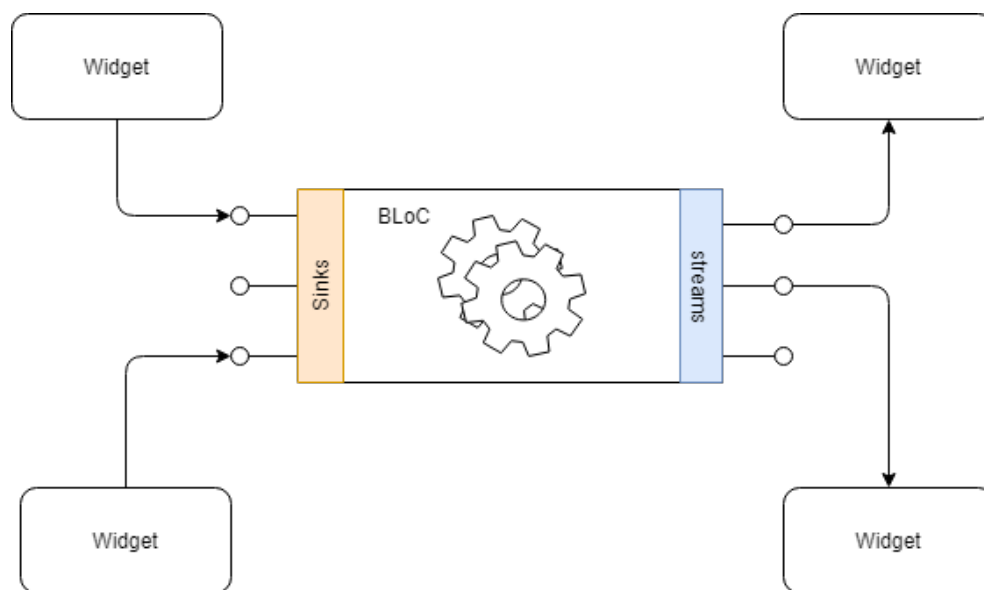


FIGURE 4.3: BLoC Architecture

4.3.1 Flutter BLoC library

I use the flutter_bloc package [2] to provide a bloc approach in my application. It is a very comfortable and easy-to-use package that offers different widgets to provide BLoC state management in Flutter applications.

Bloc Widget needed in application :

- **BlocProvider** is a widget that gives its children access to a certain block. It allows you to create an instance of a block only once, and all widgets in the tree will have access to it.

- **MultiBlocProvider** can give access to a few Blocs at the same time.
- **BlocBuilder** - we use it over the widgets that had to be rebuilt if an event has happened and the state has changed.
- **BlocListener** is working similarly. It is used when we want to do something in response to state changes. For example, in our application, we use it for navigation and showing alerts if a user enters the wrong password.

There are also a few more widgets in the flutter_bloc package, but I did not use them in this application. In the code, we can create a new bloc like a separate class that extends the Bloc class with the asynchronous **mapEventToState** function that returns Stream. This function sets bloc in the proper state depending on the event that happened.

Example of use :

```
class NewBloc extends Bloc<BlocEvent, BlocState> {
  NewBloc () : super(InitState());

  @override
  Stream<BlocState> mapEventToState(BlocEvent event) async* {
    if (event is AEvent) {
      yield AState();
    } else if (event is BEvent) {
      yield BState();
    }
  }
}
```

So every Block has an event and state. The bloc state can be represented as a separate class. It can also be an abstract class with several consequences.

```
abstract class BlocState {}
class AState extends BlocState {}
class BState extends BlocState {}
```

Similarly with the event. The bloc can have one event or several different events.

```
abstract class BlocEvent {}
class AEvent extends BlocEvent {}
class BEvent extends BlocEvent {}
```

Then Widgets that are provided with this Bloc would be able to put events to it :

```
context.read<NewBloc>().add(AEvent());
```

And widgets wrapped with **BlocBuilder** will react to the state changing. In figure 4.4, we can see what this workflow looks like :

- When a user triggers some UI element, it sends an event to the Bloc.
- Bloc reacts to this event, it can ask for some data from the repository.
- Then Bloc changed his state.
- And the UI element rebuilds itself in response to state changing.

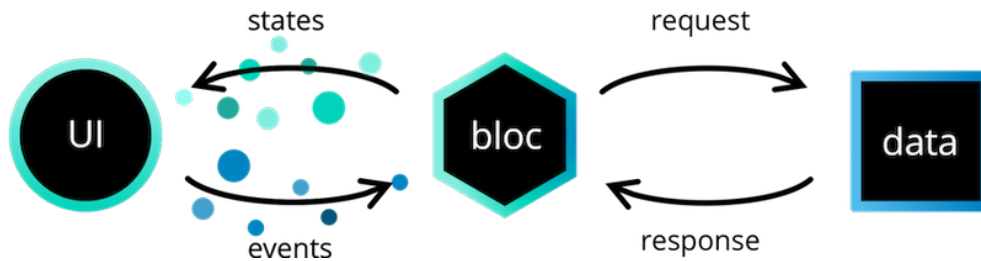


FIGURE 4.4: BLoC Architecture Workflow

4.4 Repository

A repository is a pattern that encapsulates the logic of working with data sources. The repository pattern is used for communication with the REST API. In the repository, data that comes in JSON can be turned into models. Those models can be used by widgets and blocks in the application. The repository can be introduced as a class with methods that return some data.

In our case repository must implement the methods :

- login(email, password) - method for authentication with checking user login and password, and if his reader's ticket is not expired.
- logout()
- search(query) - method returns a list of books with titles or authors that match the search word.
- recent() - returns list of 20 newest books.
- borrowed() - returns a list of borrowed books for a specific user.

4.4.1 MySQL Repository

It is common for mobile applications to use API to communicate with databases. However, Koha Rest API does not provide the necessary methods. Thus, my first approach to reaching the data was a direct connection to the database. In that case the project's workflow would look like in the figure 4.5.

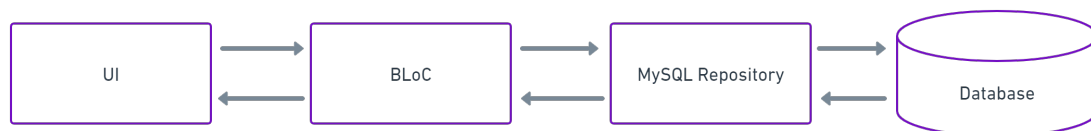


FIGURE 4.5: Workflow with MySQL Repository

To open a connection to the database from the source code, I needed:

- Host
- Port
- User
- Password

But since the port was closed, it was impossible to connect to the database from the code. Before connecting to the database, I needed to open a connection to the server via SSH. I did it with PuTTY using the SSH Tuning method. It is described in more detail in the section [2.1.4](#).

I use the `mysql` package[11] to open a connection to the MySQL database in the Dart code. There is an example below, how it looks like :

```
Future<MySQLConnection> getConnection() async {
  var settings = ConnectionSettings(
    host: '10.0.2.2',
    port: 3307,
    user: 'user_name',
    password: 'user_password',
    db: 'db_name',
  );
  return await MySQLConnection.connect(settings);
}
```

The code above shows one of the repository methods for opening the connection. It is important to note that instead of localhost address 127.0.0.1, I use 10.2.2.2 to connect to the local web server. Because in this case, 127.0.0.1 will refer to the emulator itself and not to the server.

In the repository, we also have all necessary methods: `login()`, `search()`, `recent()`, and others. They run proper SQL queries with the `query()` method from `mysql` and return data fetched from a database or null if nothing has been found.

However, this approach did not satisfy us because opening such a connection required third-party software and port redirection. Therefore, the application will not be able to work independently and access the database because the server has a closed 3306 port.

4.4.2 API Repository

API is a much better, secure, and more common way for mobile applications to obtain data. This approach became possible after the implementation of API with Python and Flask and its deployment on the Apache server. More about API implementation in the chapter [5](#).

In that case projects workflow would look like on the figure [4.6](#).

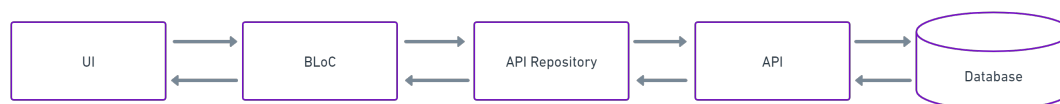


FIGURE 4.6: Workflow with API Repository

API Repository is a separate class that encapsulates communication with the API. It implements methods : `login()`, `logout()`, `autologin()`, `search()`, `recentBooks()`, `borrowedBooks()`.

I use `http` dart library[4] to make requests to the API.

```
Uri.http(baseUrl, path, queryParameters)
```

Repository methods :

- **login()** method accepts the user's email and password and checks whether such a user exists. It requests the API and gets data in JSON. If the response code is 200, the user is found, and the method returns true.

The method throws an error if the response code is 403 means that the user is not in the database, the user has not got a reader's ticket, or the reader's ticket expired.

Method stored user's email and password in the secure storage, more details about secure storage in section [4.4.4](#).

- **logout()** method deletes user data from the secure store.
- **autoLogin()** method works similarly to the login method but with some differences. It takes the user login and password from the secure storage. It returns true if the user was successfully logged-in and false if there was no user data, the response code is not 200 or if the reader's ticket expired. Throw an error if there is no Internet connection.
- **search()** method accepts the query parameter, makes an HTTP request, and returns a list of books that matches the query.
- **recentBooks()** method returns a list of the 20 newest books in the library. Returns an empty list if the response code is not 200.
- **borrowedBooks()** method returns the list of borrowed books for the specific user. It also converts the book's `deu_date` value to `DateTime` type. Later it will be used on a borrowed books screen to make expired books' cards red

4.4.3 Service locator

In our application, Business Logic is separated from UI. However, it is still possible that access to the repository will be required from both the UI and the blocks. Creating a repository class instance every time would be a bad practice. To avoid this, we can use a provider, but in such a case, we will need `BuildContext` to get to the repository from the business layer.

In this project I use `get_it`[9]. "This is a simple Service Locator for Dart and Flutter projects with some additional goodies highly inspired by Splat. It can be used instead of `InheritedWidget` or `Provider` to access objects e.g. from your UI." [9]

Advantages of using `get_it`:

- Opportunity to create an instance of `service(repository)` only one time.
- We can easily access repositories from everywhere. Business logic can be separated into the blocs, but there may be a situation when the developer needs to reach the repository from the UI.
- The library does not complicate our widget tree, because it does not use any provider widgets. And it also does not rely on `BuildContext`.
- The package is easy to use and it works fast.

Example of using :

In the `locator.dart` file :

```
final locator = GetIt.instance;

void setUp() {
  locator.registerLazySingleton<ApiRepository>(() => ApiRepository());
}
```

In the main() function :

```
void main() {
  setUp();
  runApp(const MyApp());
}
```

Access repository instance via locator anywhere in code :

```
locator.get<ApiRepository>().login(email, password)
```

4.4.4 Secure Storage

The user's email and password are stored with the flutter_secure_storage to provide the opportunity of auto login in the future - "A Flutter plugin to store data in secure storage: use Keychain² is used for iOS AES encryption is used for Android. AES secret key is encrypted with RSA and RSA key is stored in KeyStore³"[10]. The plugin provides the opportunity to store user data between program launches.

Storage is performed in a separate class with methods to read from storage and write to storage on both iOS and Android platforms. Thus, the user does not need to enter his email address and password every time he opens the application. The application will do it. If the user logged out, his data will be deleted from the storage.

4.5 Navigation

This application has six different screens. So it should be possible to switch between them and return to the previous screens, maintaining their state. In addition, during switching, you may need to transfer the information to the next screen. It can be implemented with the Flutter Navigator⁴ class.

Navigator is a widget that provides stack-like navigation. That means that when we visit a new screen, the screen's route is pushed into the stack. When we go back, we make a pop. The element at the top of the stack deletes, and the new top screen appears. See figure 4.7.

Simple use of pop and push methods from Navigator can make code messy and create a lot of duplications.

To solve this problem, we can define navigation routes, which means that the application should visit a specific route using the pushNamed() method when the navigator is called.

So instead of doing this :

```
Navigator.of(context).push(MaterialPageRoute(builder: (context) =>
  NewScreen()));
```

²https://developer.apple.com/documentation/security/keychain_services#//apple_ref/doc/uid/TP30000897-CH203-TP1

³<https://developer.android.com/training/articles/keystore>

⁴<https://api.flutter.dev/flutter/widgets/Navigator-class.html>

we do this :

```
Navigator.of(context).pushNamed("/new_screen_route");
```

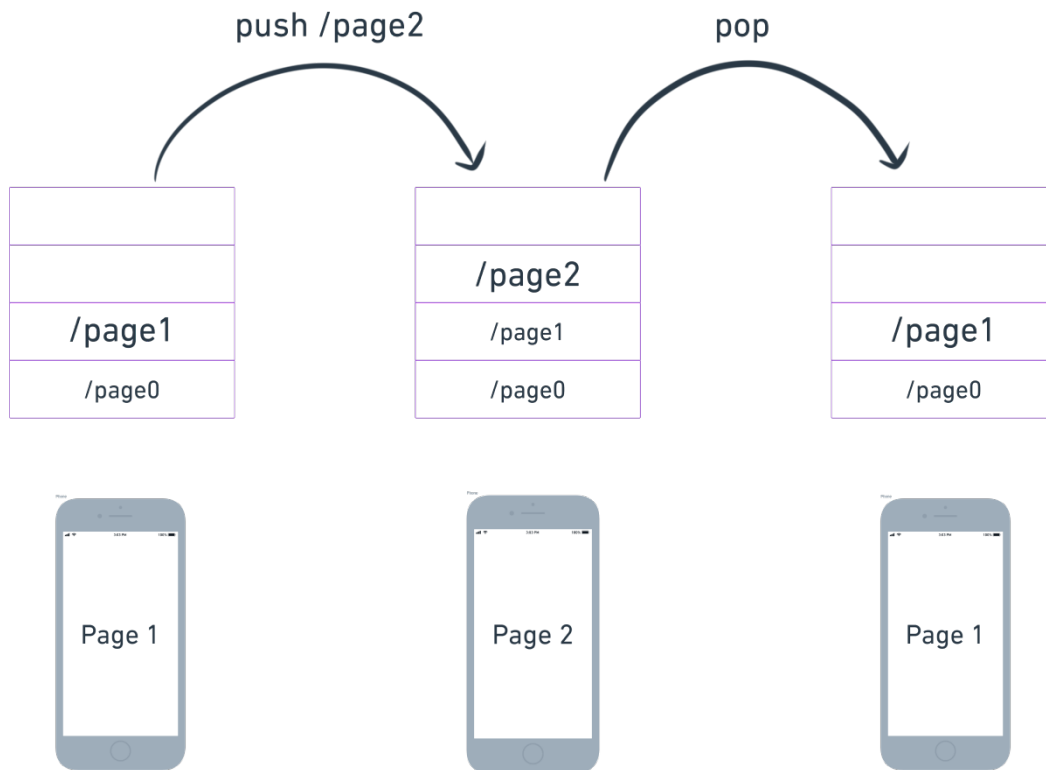


FIGURE 4.7: Navigation

All routes should be defined in the router property of the MaterialApp class.

In my case, I use **onGenerateRoute**, because by using the **onGenerateRoute** property, I can send arguments to the Screen widget, which is impossible with the simple routes property.

Code example :

```
initialRoute: "/",
onGenerateRoute: (settings) {
  if (settings.name == "/"){
    return MaterialPageRoute(
      builder: (context) => const SplashScreen());
  } else if (settings.name == "/login") {
    return MaterialPageRoute(builder: (context) => LoginScreen());
  } else if (settings.name == "/book") {
    final args = settings.arguments;
    return MaterialPageRoute(
      builder: (context) => BookScreen(book: args as Book),
    );
  }
}
```

Then if I want to navigate to the book's screen I can do something like this :

```
Navigator.of(context).pushNamed("/book", arguments: book);
```

Where a book is a book model instance.

4.6 Screens

This section explains how screens have been implemented and what users can do with them.

4.6.1 Splash Screen

Splash Screen is an initial screen of the application. It performs approximately for a second after you open the application, then it switches to another screen. The main task of this initial screen is to decide what screen should be open next: The Login screen or the Home screen. The Business Logic of this screen is presented in the Session block. The Session block has only one event, `AutoLoginEvent`. If this event happens, the bloc accesses the API repository and calls the `autoLogin()` function. If the user is active, the bloc's state changes to `SuccessAuthState`, which means that the user is logged-in. If the user does not exist, the state becomes `FailedAuthState`, which means that the user is not logged-in. If there is no internet connection, then the state becomes `NoInternetConnectionState`. The Splash screen widget uses `BlocListener` to listen to whether there are any changes in the state of the session block.

Accordingly,

- If the state is `SuccessAuthState`, and the user is logged in, we navigate to the Home screen.
- If the state is `FailedAuthState` and the user is not logged in, we navigate to the Login screen.
- If there is no internet connection, a popup appears asking you to check the internet connection.

The important thing is that when we switch from a Splash screen to another screen, we no longer have to return to the Splash screen. So instead of using the `pushName()` function from the navigator, we use `pushReplacementNamed()`. This function allows you to replace the previous route after specifying a new one. There was no more this screen route in the router stack. See the difference in the table 4.1.

When, in the future, we will make a `pop()` method, this route will no longer be in the navigator stack. Thus, If we push the "back" button on the Home screen or Login screen, we will get out of the application.

<code>pushReplacementNamed()</code>	<code>pushNamed()</code>
	<code>/home_screen</code>
<code>/home_screen</code>	<code>/splash_screen</code>

TABLE 4.1: Navigator push functions' stacks comparison

Screen shows the logo of UCU Library. See figure 4.8.



FIGURE 4.8: Splash Screen

4.6.2 Login Screen

If the user is unauthenticated, after the Splash screen, he will visit the Login screen. The user can also enter this screen by pressing the logout button on the Home screen.

The user can enter his email and password to log in, or if he wants to enter an application without authentication, he can press the gray button at the bottom of the screen. See figure 4.9.

This screen uses a login bloc. Login bloc has a few events :

- EmailChangeEvent - happens when a user inputs something in the email text field.
- PasswordChangeEvent - occurs when a user inputs something in the password text field.
- HidePasswordEvent - happens when a user clicks the eye button to hide or view his password. See figure 4.10.
- SubmitEvent - happens when a user presses the green submit button.

Login bloc has one state with different properties, and when any event happens, the bloc changes the corresponding property inside the state. I use the copyWith() function to provide it.

```
class LoginState {  
    final String email;  
    final String password;  
    final bool hidePassword;  
    final FormStatus formStatus;  
    ...  
}
```

All Login screen widget is wrapped with BlocProvider, which allows accessing the login bloc from this screen's widgets. I use the Flutter Form widget to create forms with validation. This Form widget uses BlocListener, and when LoginState.formStatus property is FailedFormStatus, a popup window appears with an error message, see figure 4.11. When users try to submit the wrong email format or empty email and password values, the Form shows the red messages under the text input fields, see figure 4.12.

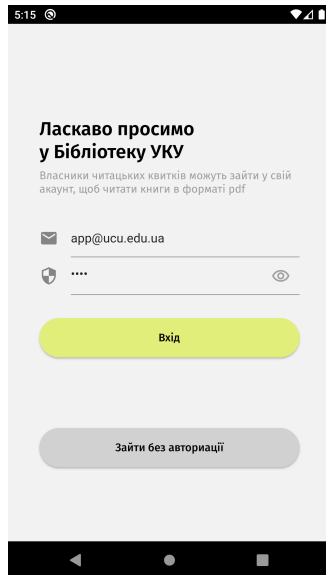


FIGURE 4.9: Login Screen

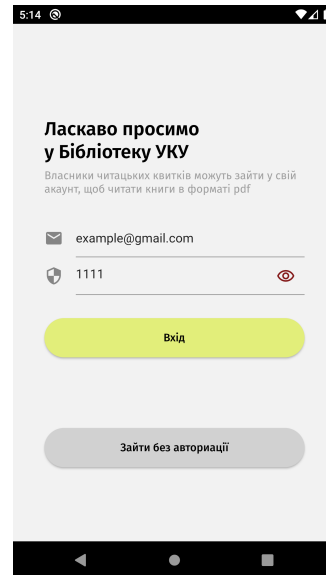


FIGURE 4.10: Login Screen show password

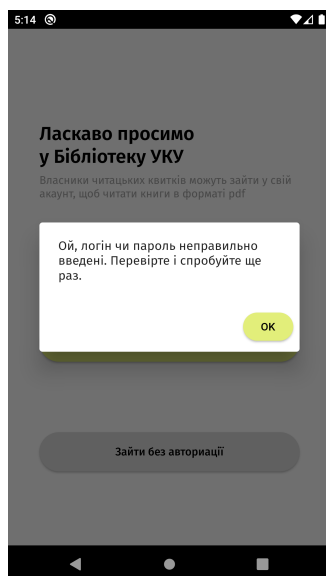


FIGURE 4.11: Login Screen alert popup

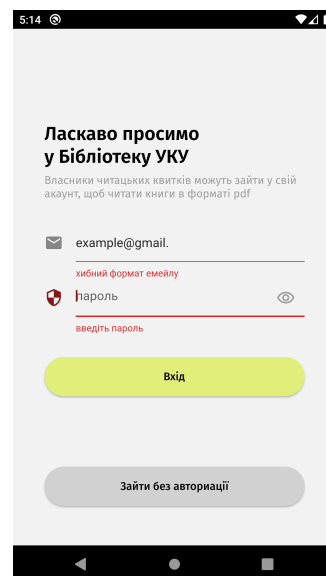


FIGURE 4.12: Login Screen validation

4.6.3 Home Screen

When a user enters the Home screen, he can see the search bar and list of recent books.

This screen widget uses a search bloc. This block has three states :

- DefaultSearchingState - if there is a default state, you can see a list of recent books on the screen, the figure 4.13.
- CurrentSearchResultsState - screen shows searching results, the figure 4.14.
- LoadingState - screen shows a progress indicator, the figure 4.16.

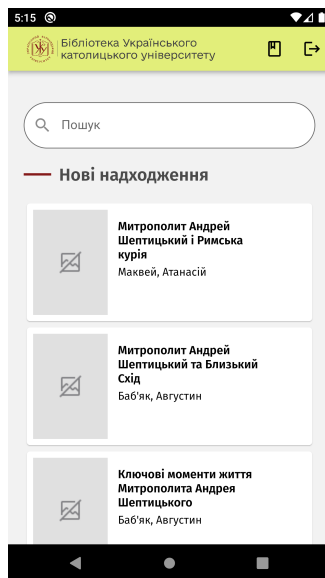


FIGURE 4.13: Home Screen

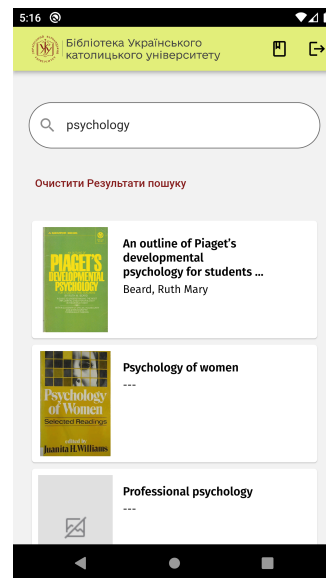


FIGURE 4.14: Home Screen, search results

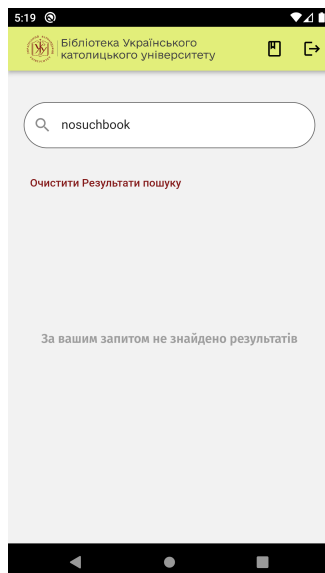


FIGURE 4.15: Home Screen, no results

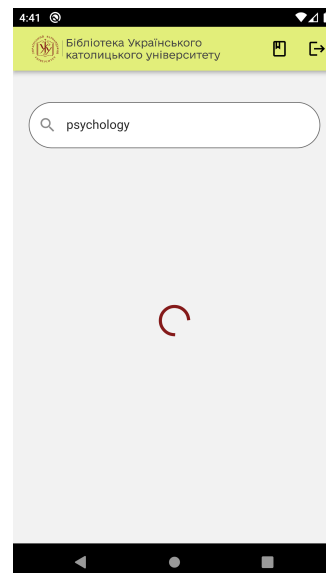


FIGURE 4.16: Home Screen, loading

When a user enters something in the search field and presses the search button, the bloc receives SearchinEvent. Then it calls a search() function from the repository

that returns a list of searched books. Then Bloc changes its state on `CurrentSearchResultsState`, and the book list can be accessed in the UI via the state's property. If there was no search result and the list is empty, the widget will display a message that there were no search results, can be seen on the figure 4.15.

If the user wants to return to the default page state, he can press the "clear search results text button" under the search bar. That will send `ToDefaultSearchEvent` to the `SearchBloc`. Then the bloc will call the `recent()` function from the repository and change its state to `DefaultSearchingSate`. A list of recent books also can be accessed via the state in the UI.

4.6.4 Borrowed Books Screen

If the user is logged-in, he can visit the Borrowed Books screen. Here, he can find the list of books he borrowed and see some information about them, see figure 4.17. For instance, he can find out when he should return the book. If some books' return date is expired, then the card with this book will be red, see figure 4.18.

The business logic of this screen is provided in a separate block. It calls a function from the repository that returns a list of borrowed books.

Bloc has three states :

- `LoadingState` - the progress indicator appears on the screen when data from the database is loading.
- `NoBorrowedBooksState` - is a state where the user has no borrowed books.
- `ListBorrowedBooksState(books : List<BorrowedBook> books)` - is a state that contains a list of borrowed books. Using this state property, the list can be accessed in the UI.

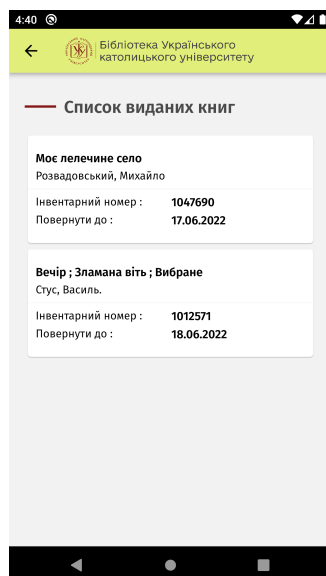


FIGURE 4.17: Borrowed Books screen

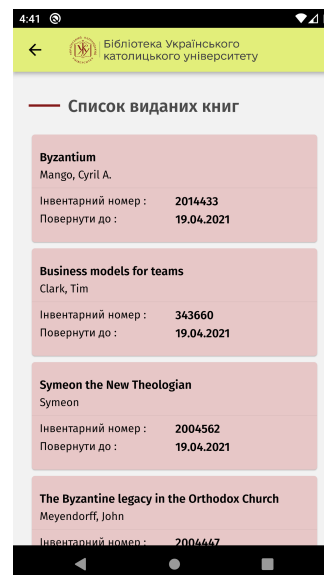


FIGURE 4.18: Borrowed Books screen, expired books

Book data stored in the separate Borrowed Book model :

```
class BorrowedBook {
  final String? author;
  final String? title;
  final String? barcode;
  final String? date_due;
  final bool? overdue;
  ...
}
```

4.6.5 Book Screen

You can access the book's screen by clicking on the book card on the Home page. The widget does not use bloc because all the information will be transmitted here by the Navigator from the previous screen.

If the user is authorized, he will be able to see the reading button at the bottom of the screen. If the book is available for reading, the button will be green. See the figure 4.19. If the book is not available in PDF, the button will be gray with a corresponding title. See the figure 4.20. Unauthorized users cannot see the button.

On this screen, the user can see :

- Book cover
- Title
- The author
- Description
- Type
- Location
- Storage code
- State, whether the book is available in the library.

If any of these values are missed in the database, it will be the string "—" instead.

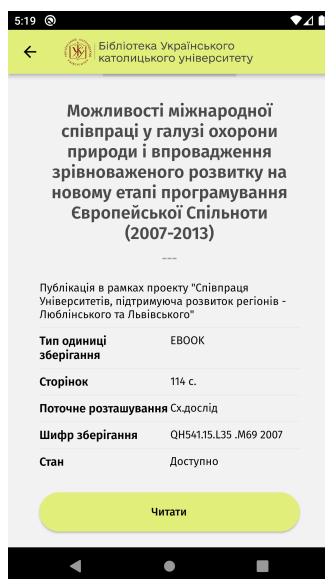


FIGURE 4.19: Books screen, reading available



FIGURE 4.20: Book screen, reading not available

4.6.6 Reading Screen

Users can visit the Reading book screen If they are authenticated and if the book is available in electronic format.

This screen widget does not use any bloc. It required only a URL(Uniform Resource Locator) with a PDF. The previous book screen gives the URL via Navigator.

To show the PDF book file, I use the plugin `syncfusion_flutterpdfviewer`[13]. This plugin has a function to view PDF files obtained from the web.

On this screen user can :

- Scroll pages.
- Switch pages with the arrow buttons on the footer bar.
- Zoom page.
- Search the page by number. Tap on the page indicator and input the page you want to visit. See figure 4.22.
- Users can not download files or copy text from files.

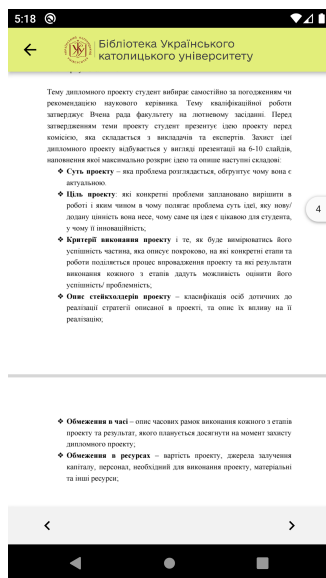


FIGURE 4.21: Reading screen

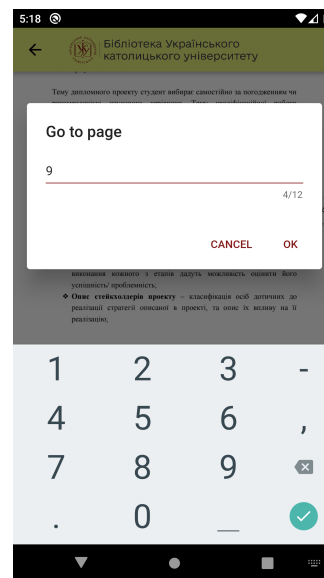


FIGURE 4.22: Reading screen, choose page

Chapter 5

API Implementation

An important part of this project development was the creation of API because, without API, the application could not function.

As mentioned in chapter 2, the Koha UCU REST API did not provide the necessary functionality for us. All these functions had to be implemented for the application to display current data from the database.

API is implemented using a Python Flask and deployed on the Apache server on the library's virtual server with the database.

Figure 5.1 shows a chart of the communication between the mobile application and the database if we use the API.

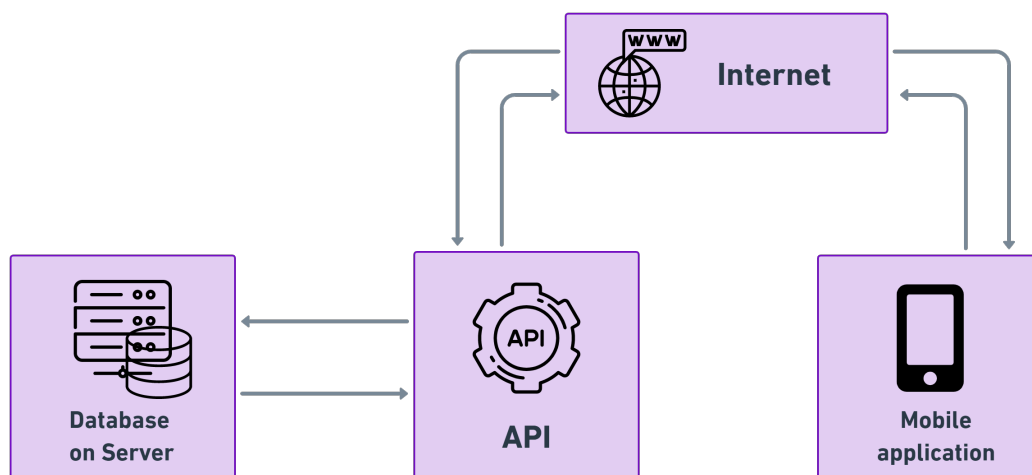


FIGURE 5.1: Communication between mobile application and database via API

During my work with the test server, the API used a not secure HTTP connection. But after migration to the production server, it uses a secure HTTPS connection.

5.1 Advantages of using API

API became the communicator between application and database on the remote server. This additional level of abstraction gives us some advantages :

- First of all, in our case, it was the only way to retrieve data from the database on the server. The direct connection was impossible and needed additional software and methods. Using API makes it easier.

- The most important advantage of API is that API makes our application more secure. If we open a connection to the database directly in the application source code, hackers can also use it. For instance, they can decompile APK files and find all necessary data. Using API can also save from SQL Injections.
- If the company decides to change the database, it will be easier to rewrite the back-end than rewrite all applications and products and then release them a few times.

5.2 Connection to the database

Since the API and the database are side by side on the same virtual server, connecting to it from the code was quite simple. To connect to the database I use the Flask-MySQL[6] extension.

I configure access to the database with such settings :

```
mysql = MySQL()

app.config["MYSQL_DATABASE_USER"] = "user_name"
app.config["MYSQL_DATABASE_PASSWORD"] = "user_password"
app.config["MYSQL_DATABASE_DB"] = "db_name"
app.config["MYSQL_DATABASE_HOST"] = "localhost"

mysql.init_app(app)
```

MYSQL_DATABASE_PORT default port is 3306, so there was no need to point it in code.

The following steps are :

1. Open connection : `conn = mysql.connect()`
2. Create the cursor to execute SQL queries and returns dictionary : `cursor = conn.cursor(pymysql.cursors.DictCursor)`
3. Execute SQL queries : `cursor.execute("Select * from example_table;")`
4. Fetch data with `cursor.fetchone()` or `cursor.fetchall()` functions
5. Finally close the connection with `cursor.close()` and `conn.close()`

5.3 Functionality

API provides access to the functionality that was necessary for the mobile application. Now, these are four get requests that do not edit anything in the database but return the necessary information from it.

Of course, in the future, the capabilities of the API can be supplemented with the update and delete methods.

5.3.1 Login

The first `/login` route aim is to check whether the user exists in the database. It takes two parameters: email and password. If they are not provided, the method returns code 400, meaning "Bad request".

```
"login?email=user@gmail.com&password=seacretpassword"
```


The function executes a SQL query to select user data from the database. If the user is found, it returns JSON with his data and if not found, then code 403 means "Forbidden".

It is also important to note that passwords in the database are hashed. Therefore, before transferring user data, the login function uses a special library to verify that password from the database and the password inputted in the application match. If the password is incorrect the function will return the response code 403. JSON contains user data like first name, last name, borrower number to take the list of his borrowed books, and others.

Response code	Response
200	JSON with user data
400	"error": "Bad request"
403	"error" : "Forbidden"
500	"error": "Internal Server Error ['error message']"

TABLE 5.1: API login responses

5.3.2 Search

The `/search` route provides a JSON file with the list of searched books' information. It requires a parameter - `q` (query). If `q` is not provided, the method returns response code 400, meaning "Bad request". The method also takes one optional parameter `pdf`, to return only books available in PDF format.

Method fetches from the database all books whose titles or author contain `q` (query). The required book's information is splitted between four tables in the database. Therefore, I use the left join in the SQL query string to get all the necessary data.

Some books have a cover image file. However, this image is stored as a BLOB (Binary Large Object). It is a special data type that can store pictures, video, or audio like binary data. Fetching such huge objects from the database and then transferring them via the internet can take a lot of time. It also can be impossible, as there is a limitation on how much data can be transferred via API response.

So to avoid transferring huge amounts of data, API transfers the link to the image instead of the huge image file. The base of the link looks like this :

```
"https://opac.ucu.edu.ua/cgi-bin/koha/opac-image.pl?biblionumber="
```

If we add the id of the book at the end, its `biblionumber`, we will have a link to the cover of this book.

```
"https://opac.ucu.edu.ua/cgi-bin/koha/opac-image.pl?biblionumber=305094"
```

Due to this, we do not need to extract BLOB files from the database at all. We can just extract the image id number from the `images` table and if it is not null, then add the `biblionumber` to a link. After that, we can add this link to our JSON response file.

JSON will contain information about searched books, like a title, author, link to PDF file, and others.

Response code	Response
200	Json with books
400	"error": "Bad request"
404	"error": "Not Found"
500	"error": "Internal Server Error ['error message']"

TABLE 5.2: API search responses

5.3.3 Recent

'/recent' route returns the newest books in the library. It has one optional parameter amount. We can define the number of recent books with it. But if we call this method without defining the amount, it returns us the 20 newest books. It returns the same information about the book as the search method. It is important to note that the list of books is in descending order. Thus, first, the user sees new books.

Response code	Response
200	Json with the newest books
404	"error": "Not Found"
500	"error": "Internal Server Error ['error message']"

TABLE 5.3: API recent responses

5.3.4 Borrowed

'/borrowed' route returns the list of books the user borrowed in the UCU Library. Methods returns books for one specific user, so it requires a user borrower number.

The method does not need to return all possible book data. It returns the book title, author, barcode, and due date. 'Due date' means till when the user should return the book to the library. So, the user will be able to find out from the application how much time has left for him to finish reading the book.

Response code	Response
200	Json with the borrowed books
400	"error": "Bad request"
404	"error": "Not Found"
500	"error": "Internal Server Error [\'error message\']"

TABLE 5.4: API borrowed responses

Chapter 6

Conclusion

6.1 Result summary

I built a mobile application for the UCU Library that allows reading e-books. Application provides all functionality that was described in section 1.3. In addition, I implemented an API that provided the mobile application with actual data from the library's database.

You can see a video demonstration of the application following this [LINK](#). On this video demonstration, application gets data from the database on the test server. That allowed me to show you all the possibilities of the application, because I have admin rights on the test server. And I can not login application that works with production server without reader's ticket. On the Github I have separate branches with test server and production server.

The Github repository of this project is private. If you need to see Github with the project's code, please contact me via email detsyk.k@ucu.edu.ua.

6.2 Future works

UCU Library application release on the Play Market and App Store is planned for the end of June.

At the next meeting with stakeholders, we plan to discuss the future of the application and possible features that will be released in the next versions of the application.

I think there will always be something to add or improve in this application. The application's functionality can be expanded depending on the needs of the library and its readers.

We can add update and delete methods to the API. This may allow us in the future, for example, to implement bookings.

We can improve the application design. Flutter allows development for the web. Therefore, if necessary, we can adapt this application to the web. Thus, the application could be used from a computer.

Bibliography

- [1] Przemyslaw Baraniak. "What is a User Flow – Everything You need to Know". In: *uxmisfit.com* (2020). URL: <https://uxmisfit.com/2020/08/17/what-is-a-user-flow-everything-you-need-to-know/>.
- [2] bloclibrary.dev. *flutter_bloc* package. URL: https://pub.dev/packages/flutter_bloc.
- [3] Didier Boelens. "Reactive-Programming-Streams-BLoC". In: *didierboelens.com* (2018). URL: <https://www.didierboelens.com/2018/08/reactive-programming-streams-bloc/>.
- [4] dart.dev. *http* package. URL: <https://pub.dev/packages/http>.
- [5] BLoC's Official Documentation. URL: <https://bloclibrary.dev/#/>.
- [6] Flask-MySQL documentation. URL: <https://flask-mysql.readthedocs.io/en/stable/>.
- [7] Flask's documentation. URL: <https://flask.palletsprojects.com/en/2.1.x/>.
- [8] Flutter documentation. URL: <https://docs.flutter.dev/>.
- [9] fluttercommunity.dev. *get_it* package. URL: https://pub.dev/packages/get_it.
- [10] *flutter_secure_storage* plugin. URL: https://pub.dev/packages/flutter_secure_storage.
- [11] Adam Lofts. *mysql1* package. URL: <https://pub.dev/packages/mysql1>.
- [12] John Ryan. "Learning Flutter's new navigation and routing system". In: <https://medium.com/flutter/learning-flutters-new-navigation-and-routing-system-7c9068155ade> (2020). URL: <https://medium.com/flutter/learning-flutters-new-navigation-and-routing-system-7c9068155ade>.
- [13] syncfusion.com. *syncfusion_flutter_pdfviewer* package. URL: https://pub.dev/packages/syncfusion_flutter_pdfviewer.