UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Coordinating Teacher:
dr. ing. Teodora Sanislav
Industrial Informatics Project

Freebris

Bejenariu Adina
Coțan Ștefania
Ferenț Sarah
Frătean Ștefania
Lupu Dima Daria

Contents

Chapter 1. Introduction

1.1. Overview

First and foremost, let us delve deeper into the meaning behind the title of our application, Freebris. The selection of the title "Free Library" is intentional and purposeful, as it reflects the core principles and objectives of our project.

The word "free" in the title emphasizes our commitment to providing unrestricted access to a vast array of literary resources. Our application aims to eliminate barriers to reading, ensuring that users can explore and engage with a diverse range of content without financial constraints.

The term "library" signifies the concept of a comprehensive collection of books, articles, and other reading materials. Our application aims to serve as a digital library, offering an extensive repository of literary works from various genres, catering to different interests and preferences.

1.2. Goals

And this leads us to our goal. We convey the idea that access to knowledge should be readily available to all. We believe that reading is a powerful tool for personal growth, intellectual development, and empowerment. Our application strives to foster a culture of continuous learning and self-improvement by promoting reading habits and facilitating easy access to valuable content

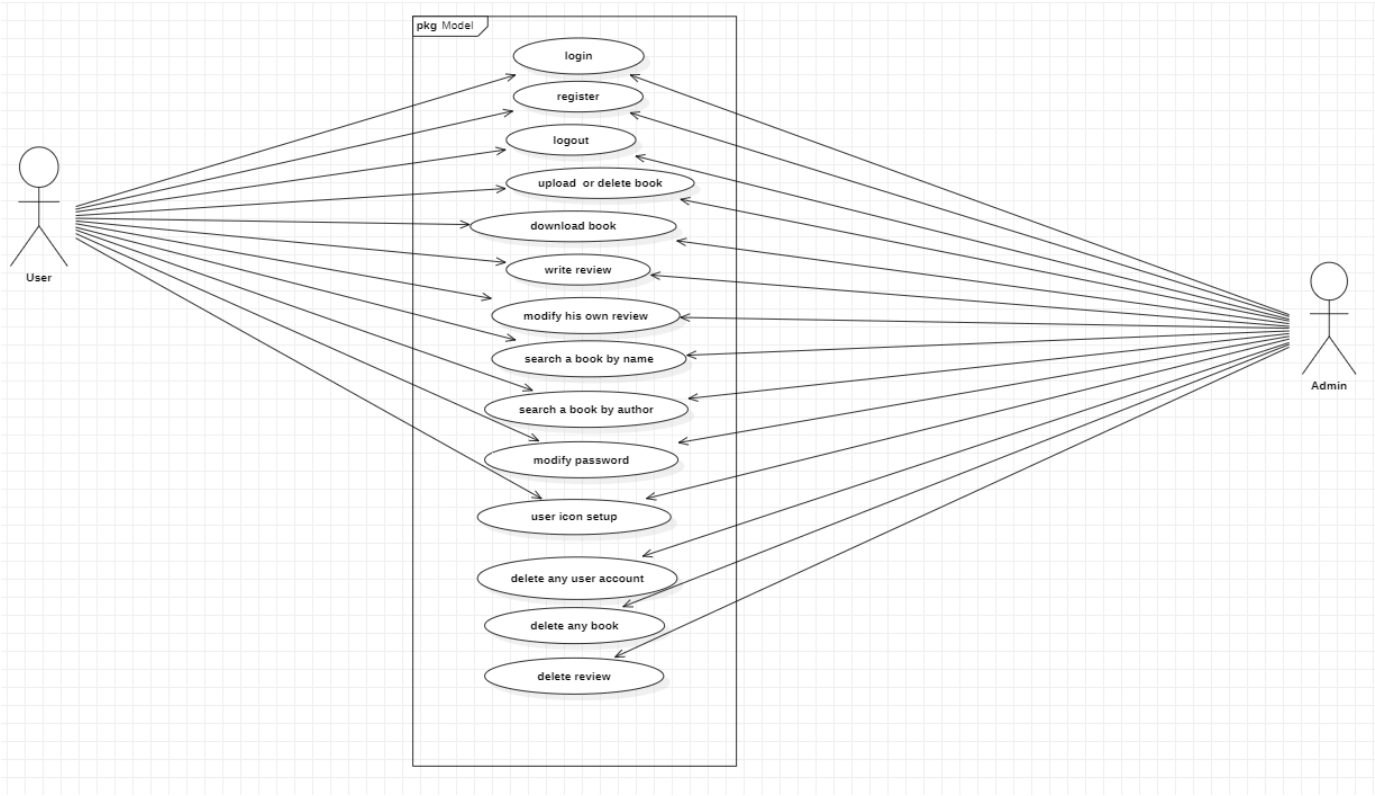
1.3. Functions

Libraries have historically been places where individuals can come together, exchange ideas, and share knowledge. In a similar vein, our application encourages social interaction among users, fostering a sense of community. Through features like discussions, book clubs, and social media integration, readers can connect, collaborate, and engage in meaningful conversations about the books they read and even share their writings.

Chapter 2. Application Design

2.1. Application Design

In order to make the logic of our application as clear as possible we're gonna attach below the use case diagram.



Our application caters to two distinct types of users: "Classic" users and "Admin" users. Each user type possesses different privileges and responsibilities within the system.

Classic users are regular users of the application who can access and utilize various features available to them. They have the following functionalities:

- **Login**: Classic users can log in to the application using their registered credentials, which include their email/username and password.
- **Register**: New users can create an account by registering with the application. They need to provide a valid email/username and a secure password to complete the registration process.
- **Upload/Delete Own Books:** Classic users can upload their own books to the application's library, making them available to other users for download. They can also delete their own uploaded books if desired.
- **Download a Book:** Classic users have the ability to browse and download books from the application's library, enriching their reading experience.
- **Write a Review:** Classic users can write and submit reviews for books they have read, sharing their thoughts and opinions with the community.
- **Search a Book by Title/Author:** Classic users can search for books within the application's library using either the book's title or the author's name, facilitating easy navigation and discovery.

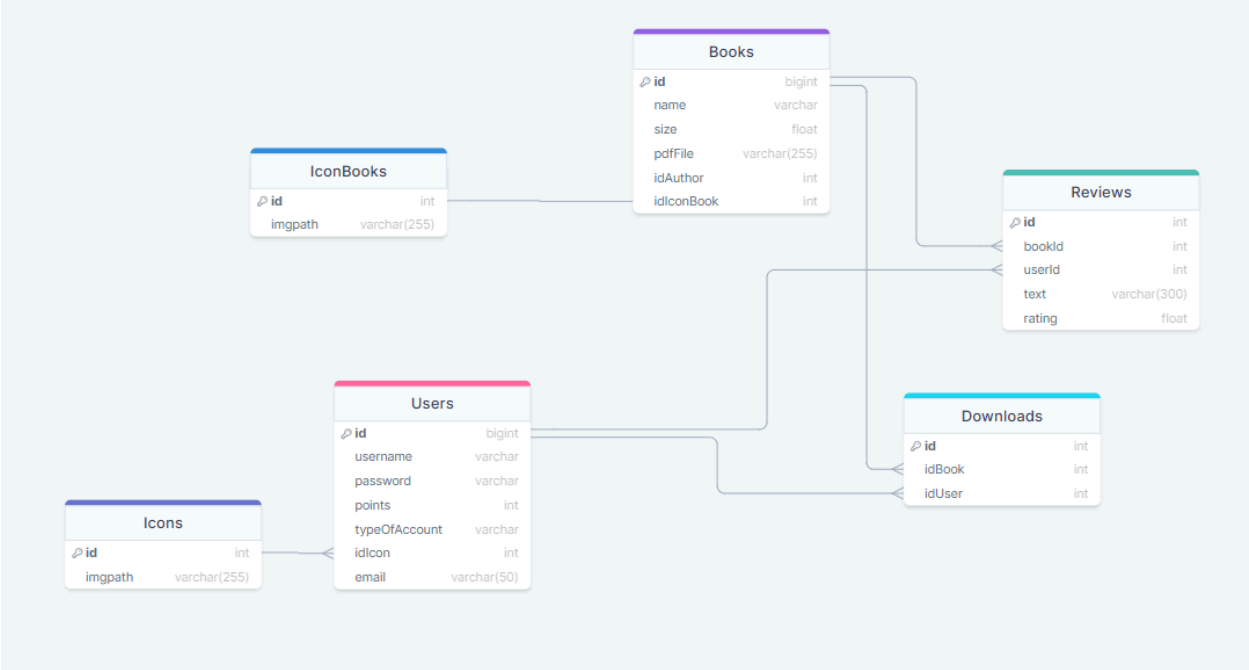
- **Modify Password, Email, and Icon:** Classic users can update their account information, including their password, email address, and profile icon, ensuring the accuracy and relevance of their user profile.
- **Delete Account:** Classic users have the option to permanently delete their account, removing all associated data and personal information from the application.

Admin users possess additional privileges and responsibilities within the application. In addition to the functionalities available to classic users, admin users have the following abilities:

- **Delete User Accounts:** Admin users can delete any user account within the application, providing them with moderation capabilities to maintain a safe and appropriate user community.
- **Delete Reviews and Books:** Admin users have the authority to delete any review or book within the application, ensuring the quality and appropriateness of the content available to users.

By implementing user types and defining their respective functionalities, our application aims to provide an engaging and secure environment for users to explore, share, and interact with reading materials while maintaining administrative control and user moderation.

2.2. Database Diagram



Database Diagram

The database contains the following tables:

- **Users**: we need this table for login or register of the users, for sending the books to their emails and the main functionalities of the application: giving comments, receiving points, and downloading or uploading books.

id	int	☐
username	varchar(20)	☐
password	varchar(255)	☑
points	int	☑
typeOfAccount	varchar(20)	☐
idIcon	int	☐
email	varchar(50)	☑

- **Books**: The books are the main object where the books are stocked and everything is connected to this table.

Column Name	Data Type	Allow Nulls
id	int	☐
name	varchar(50)	☐
size	float	☐
pdfFile	varchar(255)	☐
idAuthor	int	☐
idIconBook	int	☐
		☐

- **Icons**: are used for the icons of the users and every user can receive a different icon

Column Name	Data Type	Allow Nulls
id	int	☐
imgpath	varchar(MAX)	☐

- **IconBooks**: used for the icons of the books. Every book receives a certain icon depending on the topic the book is about.

Column Name	Data Type	Allow Nulls
id	int	☐
imgpath	varchar(MAX)	☐

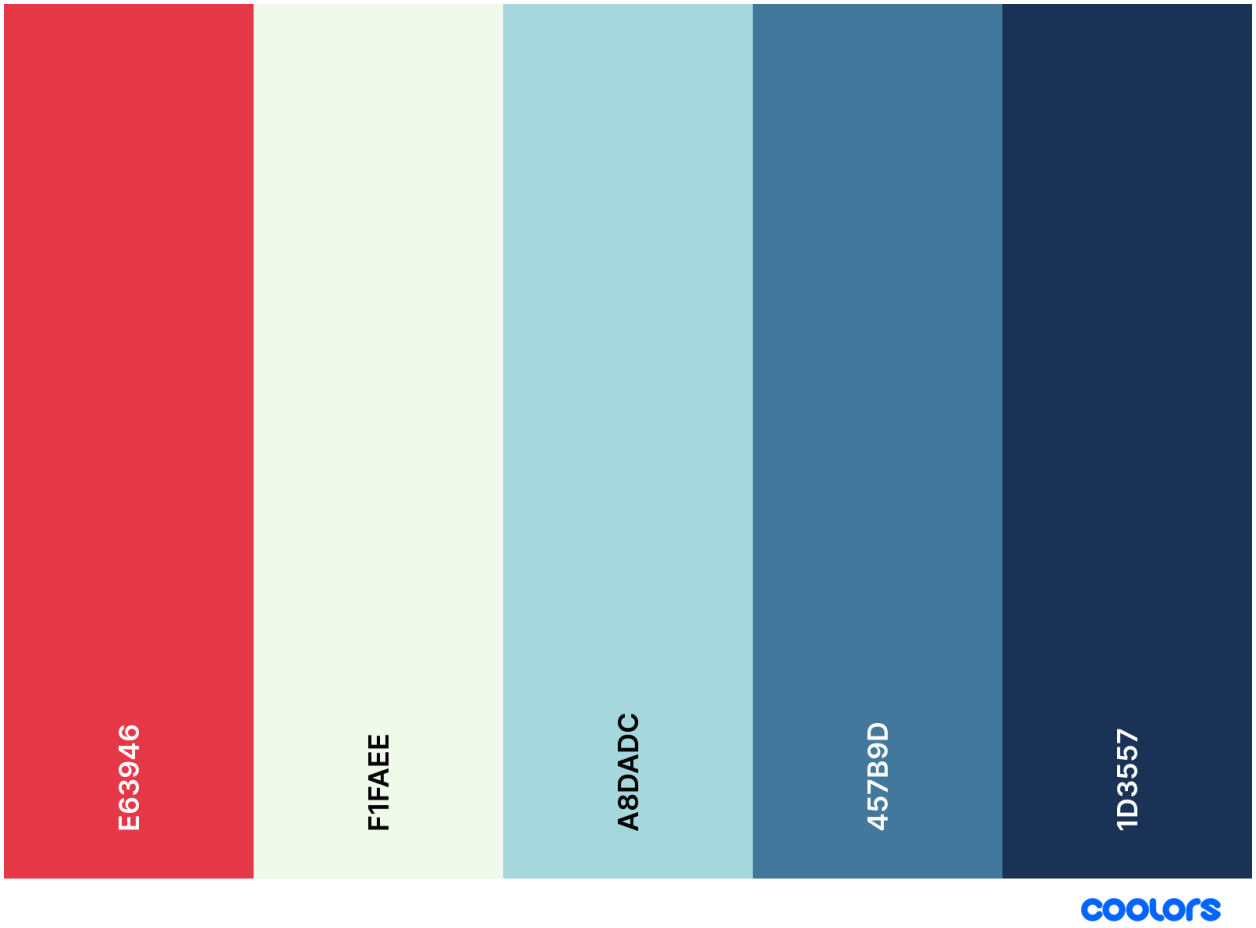
- **Reviews**: is the join table between Users and Books. This table's main usage is the way Users receive points by giving reviews to the Books.

	Column Name	Data Type	Allow Nulls
🔑	id	int	☐
	bookId	int	☐
	userId	int	☐
	text	varchar(300)	☑
	rating	float	☐

- **Downloads**: is another table that creates a join between Users and Books. This table provides the possibility for the Users to download the Books.

	Column Name	Data Type	Allow Nulls
🔑	id	int	☐
	idUser	int	☐
	idBook	int	☐

2.3. GUI design and functionality

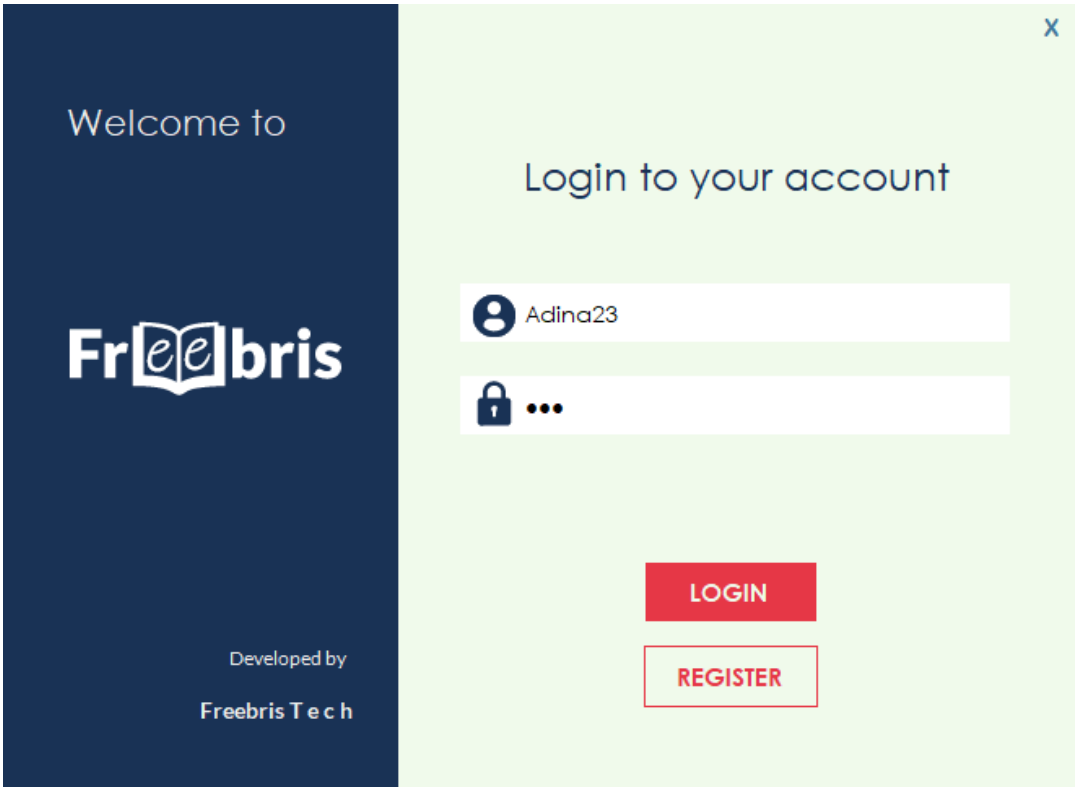


Colour Palette

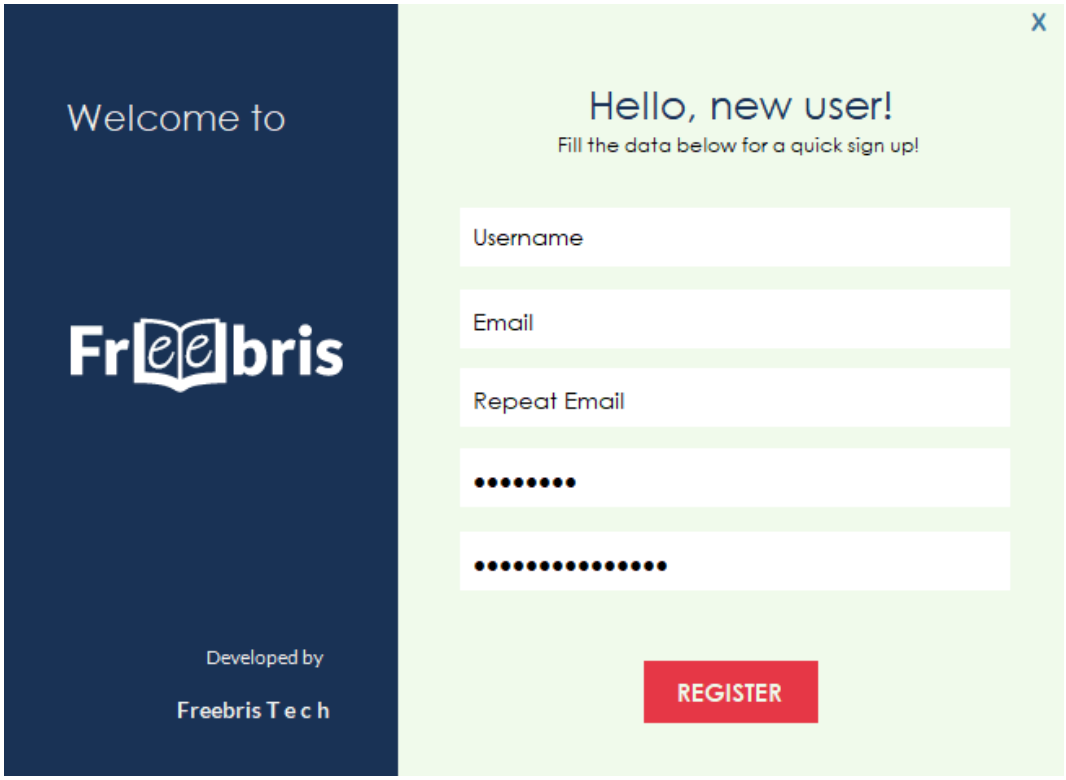
The Freebris application has two types of users: the regular user, that can download books and leave reviews and the admin, that can manage the users and the books from the app, besides the regular user functionalities. The regular user interface will be explained first, and then the additional features of the admin user.

Regular User GUI

When starting the application, the first page appearing is the **Login Page**. On this page you can Login to your account or choose to create a new account by pressing the Register button, which will open the **Register Page**.

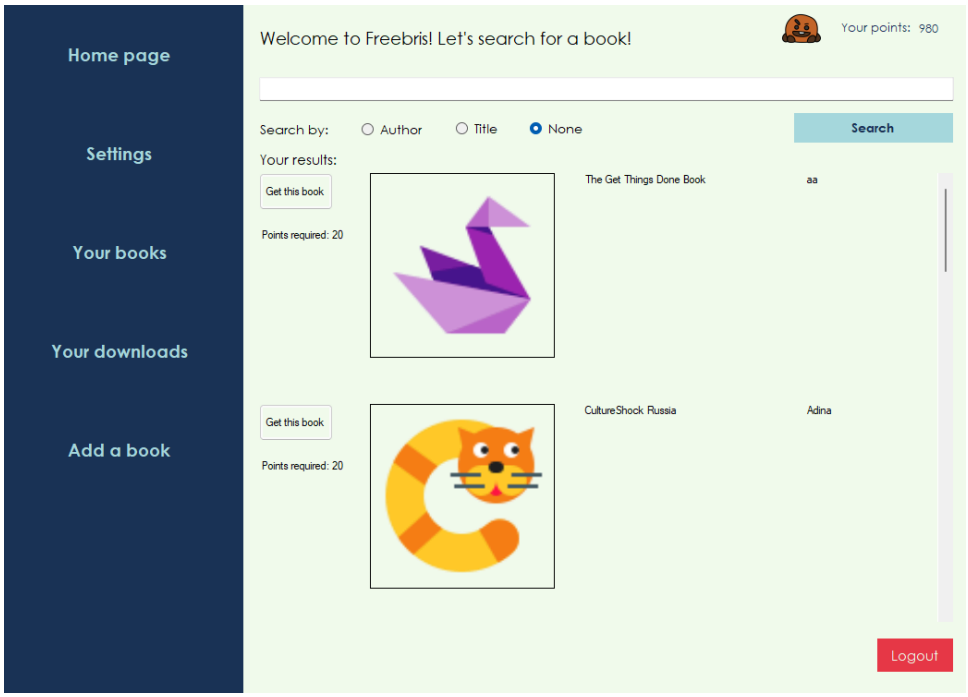


Login Page

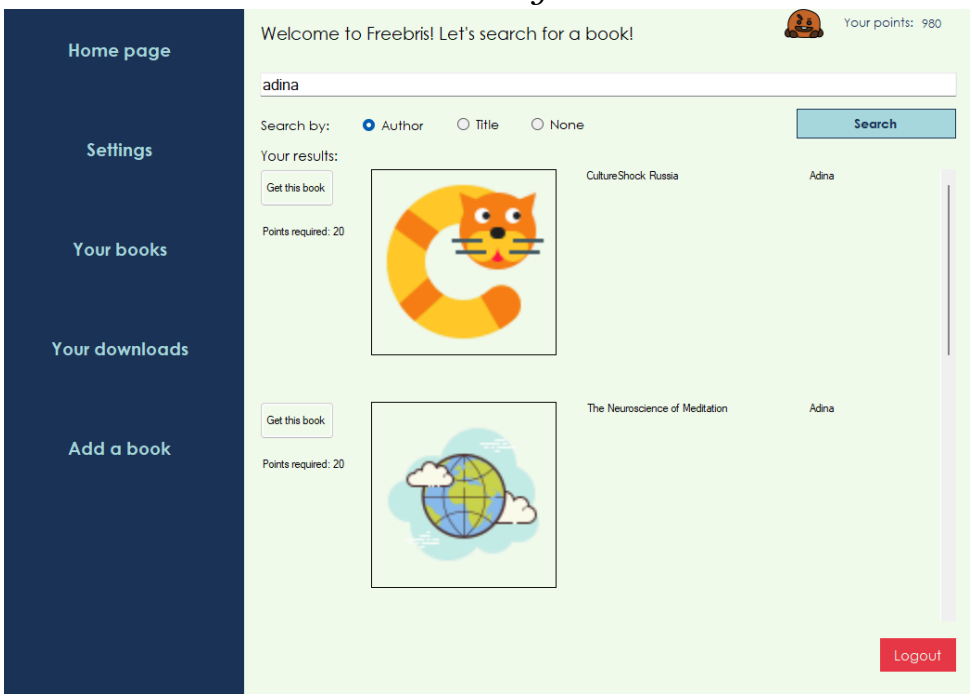


Register Page

After entering into the account, the user will be redirected to the **Home page**. On the left side is the menu which consists in the main functions that the application has: Settings, Your books - in which you can see the books the user has uploaded, Your download - in which the user can see the books that he had downloaded, Add a book - where the user can upload a new book to the app. The right side of the form will open the specified functionalities. For the Home page, here will be listed all the books in the database with a search function to easily find the wanted files. In the right corner you can see your User Icon and the number of points you have.



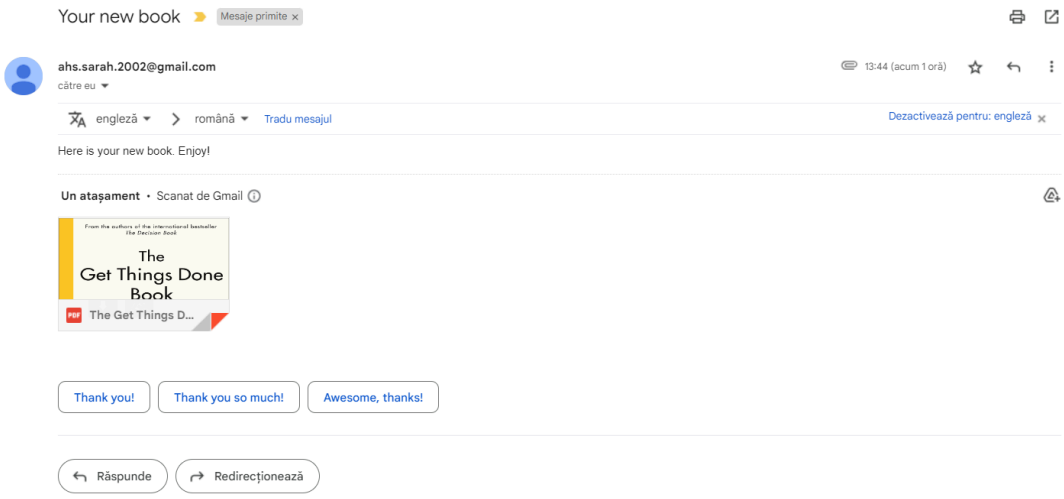
Main Page



Main Page - Search function

Freebris has a points based system for downloading books. When a new user is created, its account will receive 100 points. With these points he can download books, which cost 20 points per book. If you have less than 20 points in your account, then the download option will not work. For accumulating the points back, you have to leave reviews of the books you read, or upload new books into the application. For each review you receive 5 points and for each upload you receive 20 points back. The system was designed to encourage the Freebris Community to be more engaged in the quality of the books, by uploading books themselves, or by leaving reviews.

To download a book from the Home page the user has to click the button “Get this book”, and an email will be sent to its address with the pdf file of the book attached. To leave a review, the user has to press the book’s icon, which will open the **Review Window.** You can exit the application anytime by pressing the Logout button.

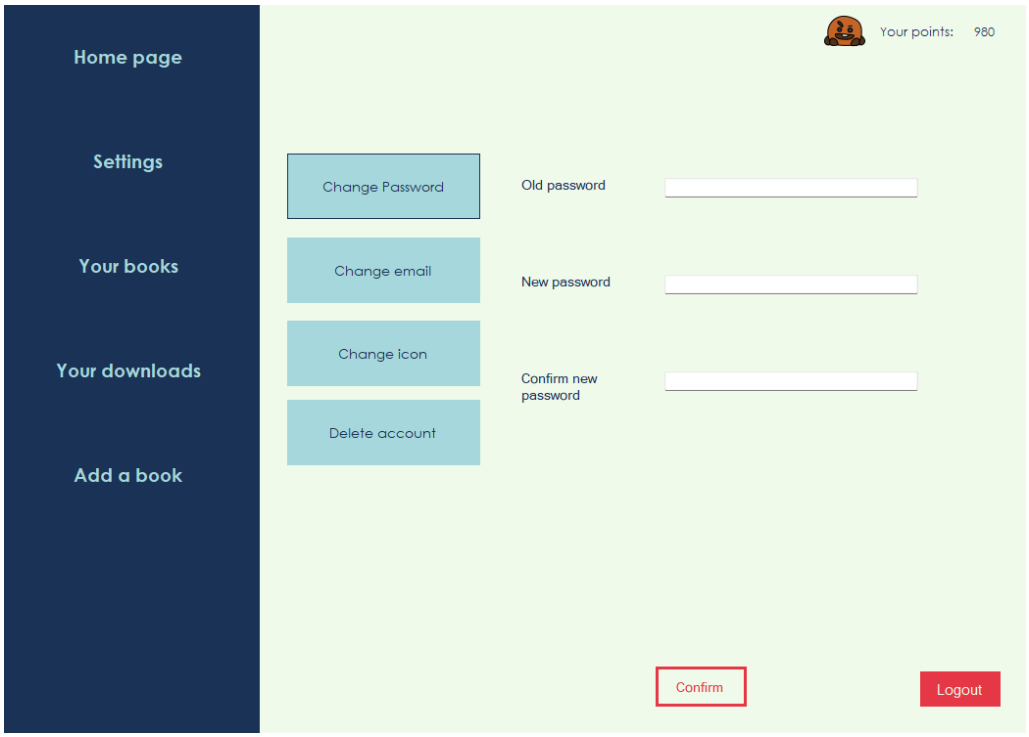


PDF sent via email

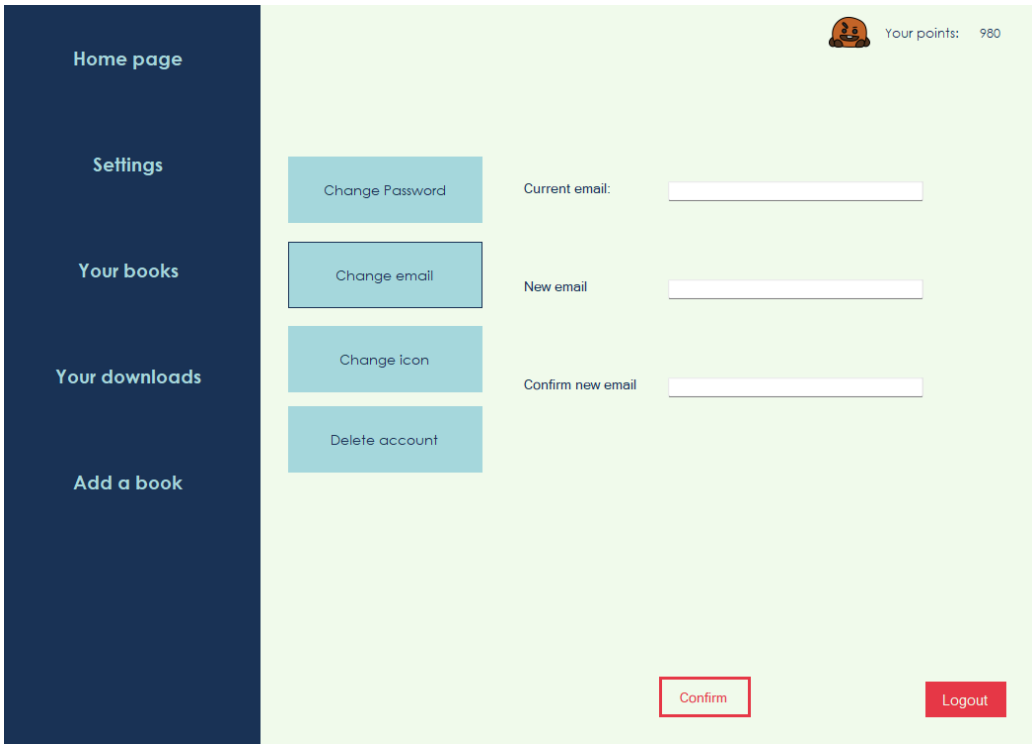
On the review pop-up you can see the previous reviews left on the book and you can delete the reviews you have previously left.

Review Page

After pressing the Settings button, the **Settings Page** will appear on the right side. Here the user can make various changes to its account - change password, change email, change icon and delete account.By pressing the corresponding button, different controls will appear on the panel in order to satisfy the need requested. The Change Password and Change Email functions work similarly. You have to input your old entry, and update and confirm it in the textboxes below.

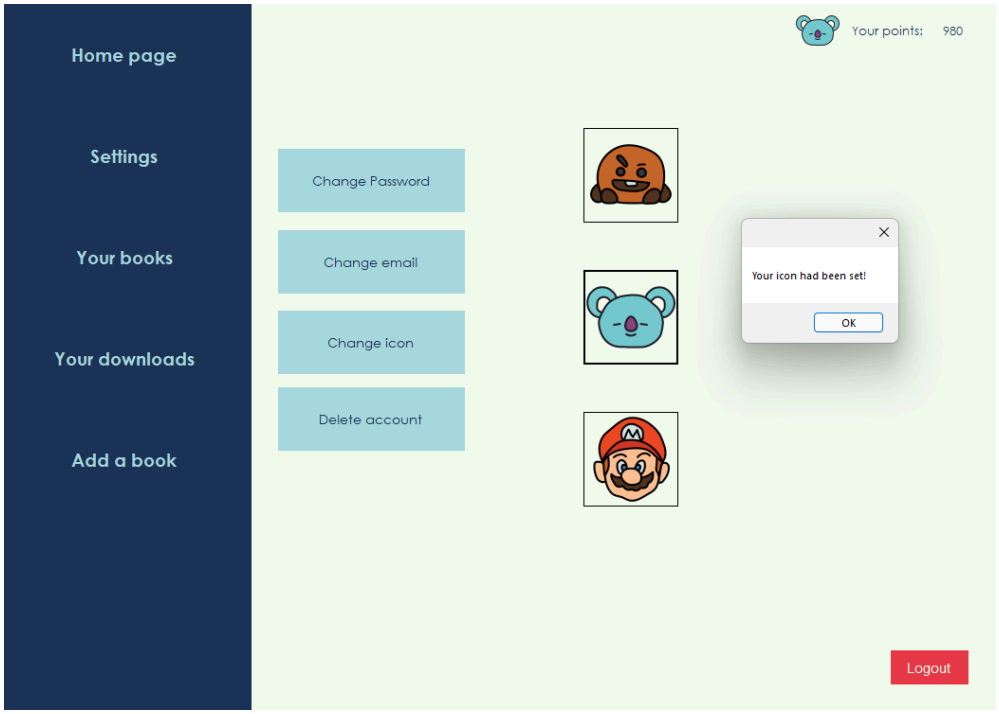


Settings Panel - Change Password



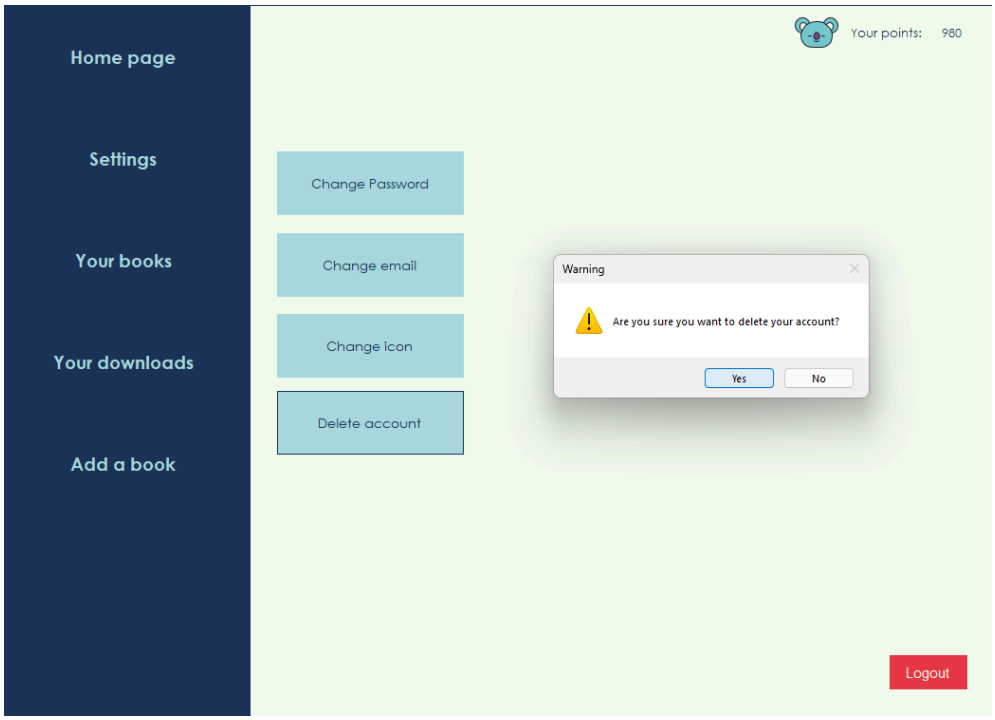
Settings Panel - Change email

Upon clicking the “Change Icon” button, the user will be greeted by the three options for the icon, and by clicking the icon you can set it as your icon.



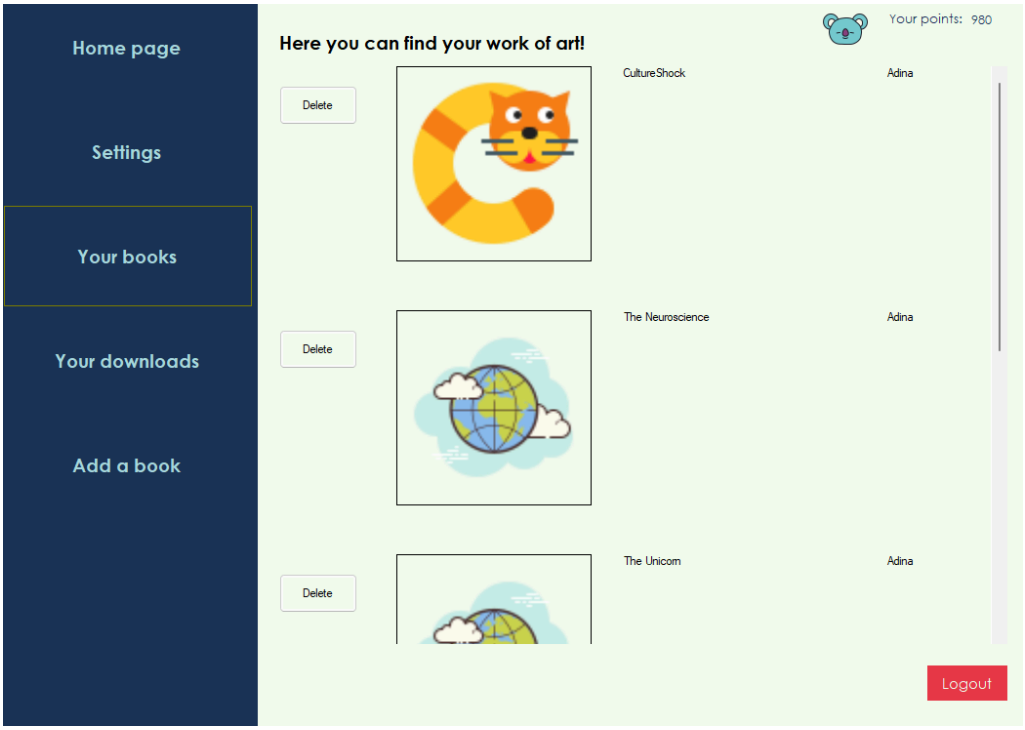
Settings Panel - Change Icon

Upon clicking the "Delete account" button, the user will be prompted with a Message Box in which he has to confirm his action. If the "Yes" button will be clicked, then the account will be deleted, otherwise nothing will happen.

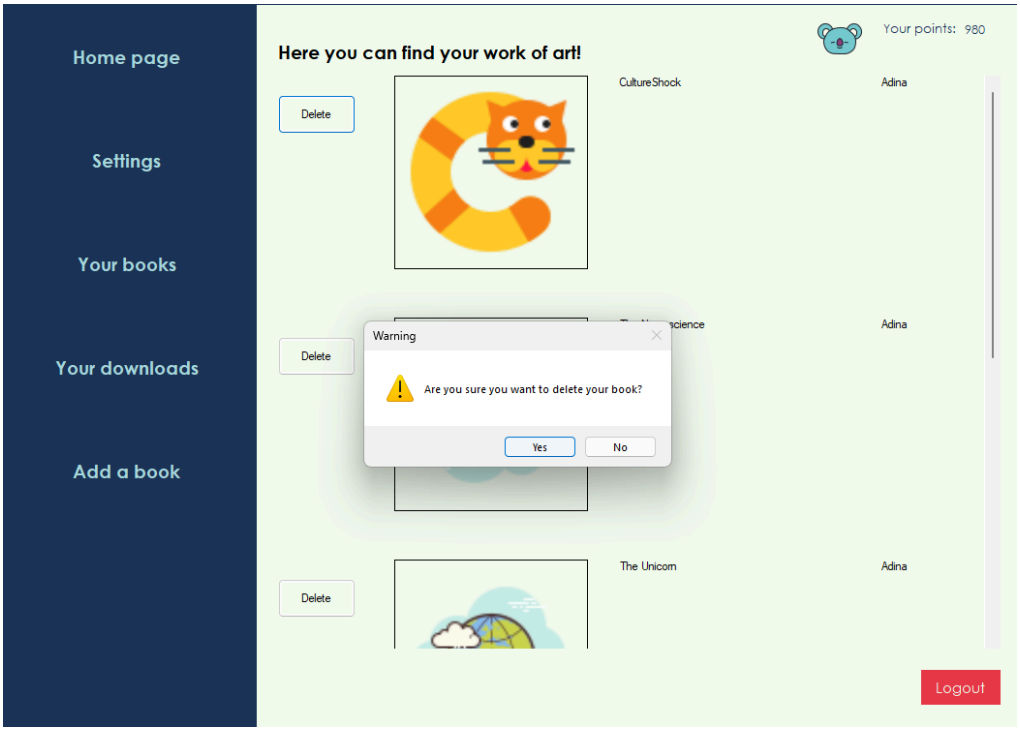


Settings Panel - Delete account

By selecting the "Your books" button, the user will get to the **Your Books Page**. Here will be listed the books that are associated with the current user (as author) and the user can choose to delete one of its books.

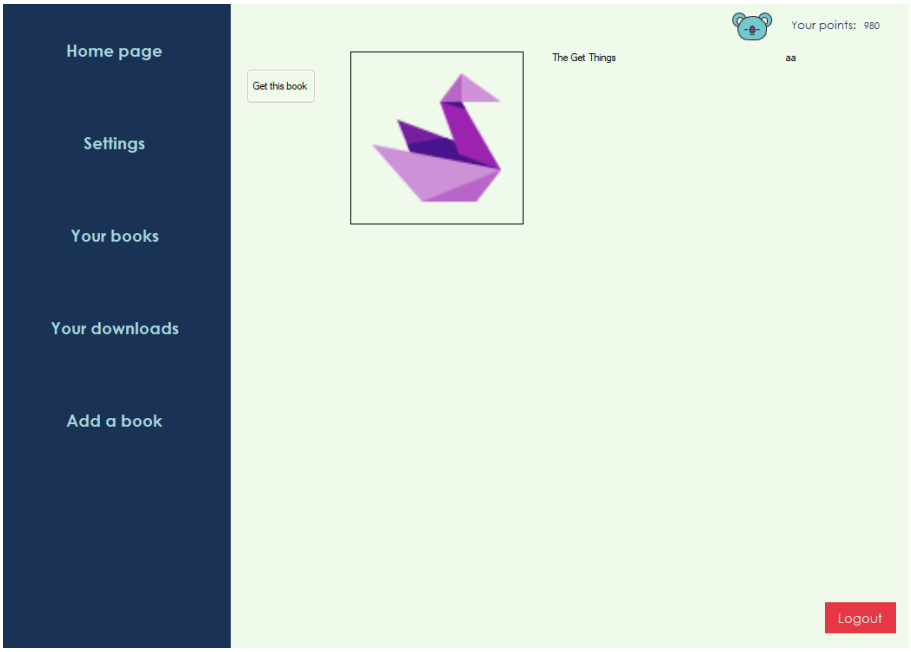


Your Books Page



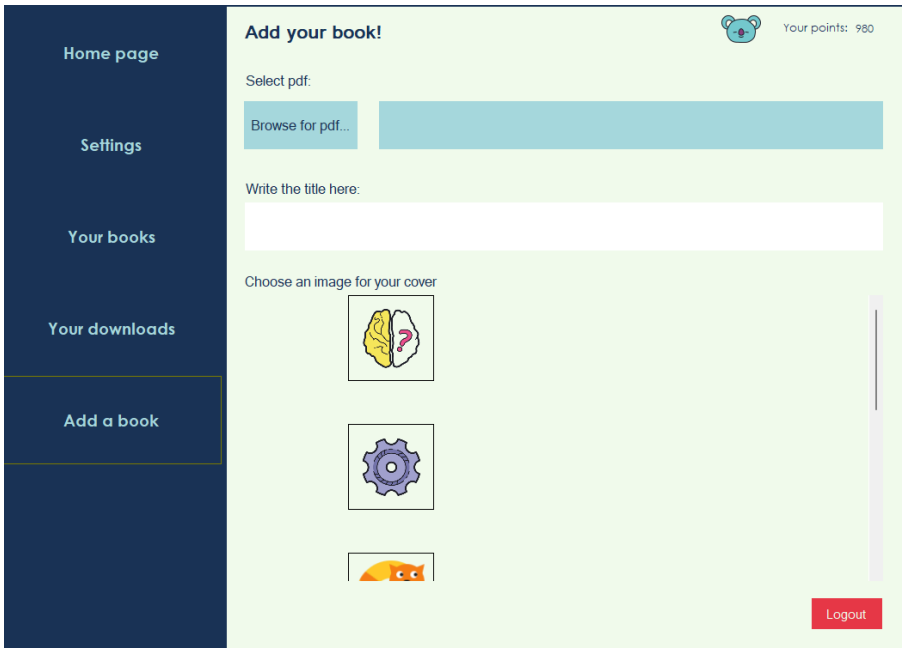
Your Books Page - Delete

The next button on the left side is the "Your downloads" button and upon clicking it the **Your Downloads Page** will load up. Here, the user will find a list with all the books downloaded from that account. In order to prevent the loss of files and points, the user can have the pdfs resent to them on their email address.



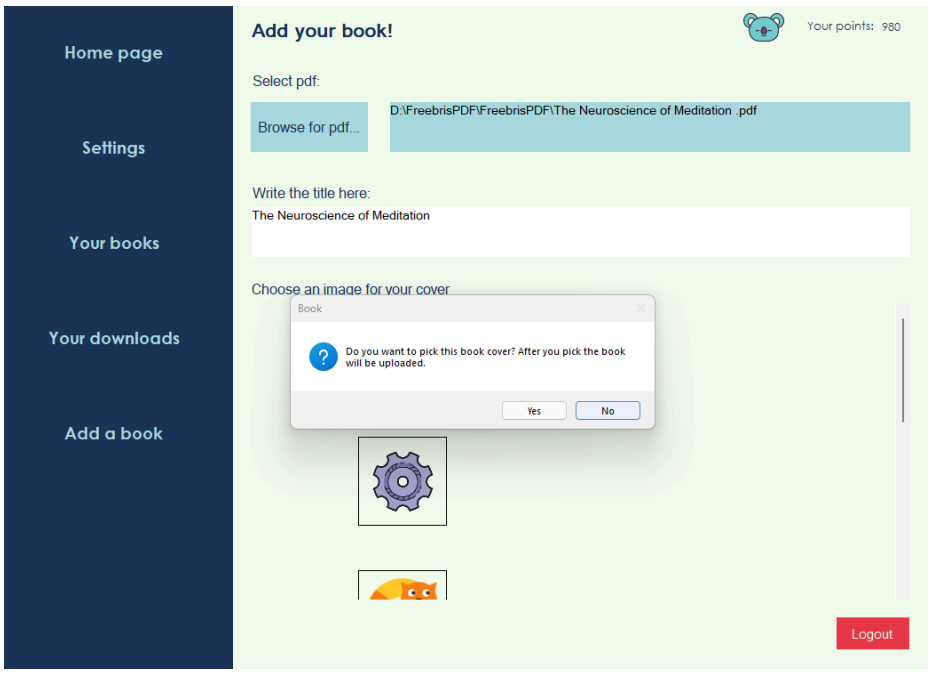
Your Downloads Page

After pressing the "Add a book" button, the **Add a book Page** will appear on the right side. Here the user can upload a new book to the app.



Add a Book Page

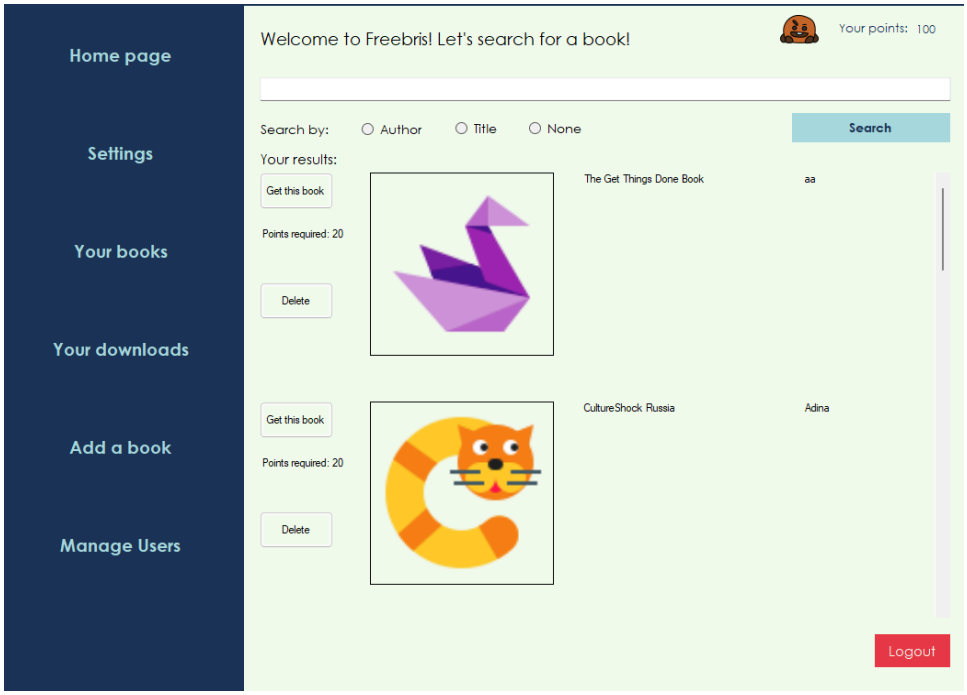
By pressing the "Browse for pdf..." button, the user can choose the file he wishes to upload. On the blue textbox next to the button, the path of the chosen file will be written. Its text cannot be modified and its purpose is to show the user exactly what file was chosen,in order to make sure that the pdf selected was the right one. On the textbox under, the user must insert the title of the book and below he will have to choose an icon. When the fields are completed and an icon is selected, a message box will appear to confirm the selections, and the book will be stored in the database.



Add a Book Page - Adding a book

Admin user GUI

The Login and Register Pages are the same for all the users, the only differences being in the extra button on the left side of the Home Page, and the delete button for each book. Only the admin users are able to see and use the hidden button “Manage users”.



Home Page - for admin

When the “Manage Users” button is pressed, the **Manage Users Page** is being displayed. Here the admin will see a list with all the users and can delete any account.

Manage Users Page

Chapter 3. Application Implementation

3.1. Work Procedure

To bring this project to life, the initial step involved creating two separate Visual Studio projects: one for the server part of the application and another one for the client part. The server application was developed as a console project, while the client application was built as a Windows Forms application.

Server Application:

- Project Type: Console Application
- Purpose: The server application serves as the backbone of the overall system, handling data processing, communication, and business logic.
- Key Features: The server application manages user authentication, book and review storage, and communication with the client application.
- Technologies/Frameworks: The server application utilizes C# and relevant libraries/frameworks for networking, data storage, and security.

Client Application:

- Project Type: Windows Forms Application
- Purpose: The client application provides a user-friendly interface for users to interact with the system's features and functionalities.
- Key Features: The client application allows users to register, log in, upload and download books, write reviews, search for books, and manage their account.

- Technologies/Frameworks: The client application is developed using C# and leverages the Windows Forms framework to create a visually appealing and interactive user interface.

By separating the server and client components into distinct projects, we could focus on implementing the specific functionalities and requirements of each part.

3.2. Database Implementation

For the database management of this project, SQL Server Management Studio (SSMS) was employed. SSMS is a comprehensive and powerful tool provided by Microsoft for managing SQL Server databases. After creating the database using SQL Server Management Studio (SSMS), the next step was to share it with other team members or users. To accomplish this, a script was generated from the database, which could be executed to create and update the database structure consistently for everyone involved in the project.

The code we used is the following one.

```
CREATE DATABASE [FreebrisDB]
ALTER DATABASE [FreebrisDB] SET COMPATIBILITY_LEVEL = 150
GO
IF (1 = FULLTEXTSERVICEPROPERTY('IsFullTextInstalled'))
begin
EXEC [FreebrisDB].[dbo].[sp_fulltext_database] @action = 'enable'
end
GO
ALTER DATABASE [FreebrisDB] SET ANSI_NULL_DEFAULT OFF
GO
ALTER DATABASE [FreebrisDB] SET ANSI_NULLS OFF
GO
ALTER DATABASE [FreebrisDB] SET ANSI_PADDING OFF
GO
ALTER DATABASE [FreebrisDB] SET ANSI_WARNINGS OFF
GO
ALTER DATABASE [FreebrisDB] SET ARITHABORT OFF
GO
ALTER DATABASE [FreebrisDB] SET AUTO_CLOSE OFF
GO
ALTER DATABASE [FreebrisDB] SET AUTO_SHRINK OFF
GO
ALTER DATABASE [FreebrisDB] SET AUTO_UPDATE_STATISTICS ON
GO
ALTER DATABASE [FreebrisDB] SET CURSOR_CLOSE_ON_COMMIT OFF
```

```
GO
ALTER DATABASE [FreebrisDB] SET CURSOR_DEFAULT  GLOBAL
GO
ALTER DATABASE [FreebrisDB] SET CONCAT_NULL_YIELDS_NULL OFF
GO
ALTER DATABASE [FreebrisDB] SET NUMERIC_ROUNDABORT OFF
GO
ALTER DATABASE [FreebrisDB] SET QUOTED_IDENTIFIER OFF
GO
ALTER DATABASE [FreebrisDB] SET RECURSIVE_TRIGGERS OFF
GO
ALTER DATABASE [FreebrisDB] SET  DISABLE_BROKER
GO
ALTER DATABASE [FreebrisDB] SET AUTO_UPDATE_STATISTICS_ASYNC OFF
GO
ALTER DATABASE [FreebrisDB] SET DATE_CORRELATION_OPTIMIZATION OFF
GO
ALTER DATABASE [FreebrisDB] SET TRUSTWORTHY OFF
GO
ALTER DATABASE [FreebrisDB] SET ALLOW_SNAPSHOT_ISOLATION OFF
GO
ALTER DATABASE [FreebrisDB] SET PARAMETERIZATION SIMPLE
GO
ALTER DATABASE [FreebrisDB] SET READ_COMMITTED_SNAPSHOT OFF
GO
ALTER DATABASE [FreebrisDB] SET HONOR_BROKER_PRIORITY OFF
GO
ALTER DATABASE [FreebrisDB] SET RECOVERY FULL
GO
ALTER DATABASE [FreebrisDB] SET  MULTI_USER
GO
ALTER DATABASE [FreebrisDB] SET PAGE_VERIFY CHECKSUM
GO
ALTER DATABASE [FreebrisDB] SET DB_CHAINING OFF
GO
ALTER DATABASE [FreebrisDB] SET FILESTREAM( NON_TRANSACTED_ACCESS = OFF )
GO
ALTER DATABASE [FreebrisDB] SET TARGET_RECOVERY_TIME = 60 SECONDS
GO
ALTER DATABASE [FreebrisDB] SET DELAYED_DURABILITY = DISABLED
GO
ALTER DATABASE [FreebrisDB] SET ACCELERATED_DATABASE_RECOVERY = OFF
GO
```

```
EXEC sys.sp_db_vardecimal_storage_format N'FreebrisDB', N'ON'
GO
ALTER DATABASE [FreebrisDB] SET QUERY_STORE = ON
GO
ALTER DATABASE [FreebrisDB] SET QUERY_STORE (OPERATION_MODE = READ_WRITE,
CLEANUP_POLICY = (STALE_QUERY_THRESHOLD_DAYS = 30),
DATA_FLUSH_INTERVAL_SECONDS = 900, INTERVAL_LENGTH_MINUTES = 60,
MAX_STORAGE_SIZE_MB = 1000, QUERY_CAPTURE_MODE = AUTO,
SIZE_BASED_CLEANUP_MODE = AUTO, MAX_PLANS_PER_QUERY = 200,
WAIT_STATS_CAPTURE_MODE = ON)
GO
USE [FreebrisDB]
GO


SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Downloads](
    [id] [int] NOT NULL,
    [idUser] [int] NOT NULL,
    [idBook] [int] NOT NULL,
 CONSTRAINT [PK_Download] PRIMARY KEY CLUSTERED
(
    [id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

/****** Object:  Table [dbo].[IconBooks]
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[IconBooks](
    [id] [int] NOT NULL,
    [imgpath] [varchar](max) NOT NULL,
 CONSTRAINT [PK_IconBooks] PRIMARY KEY CLUSTERED
(
    [id] ASC
```

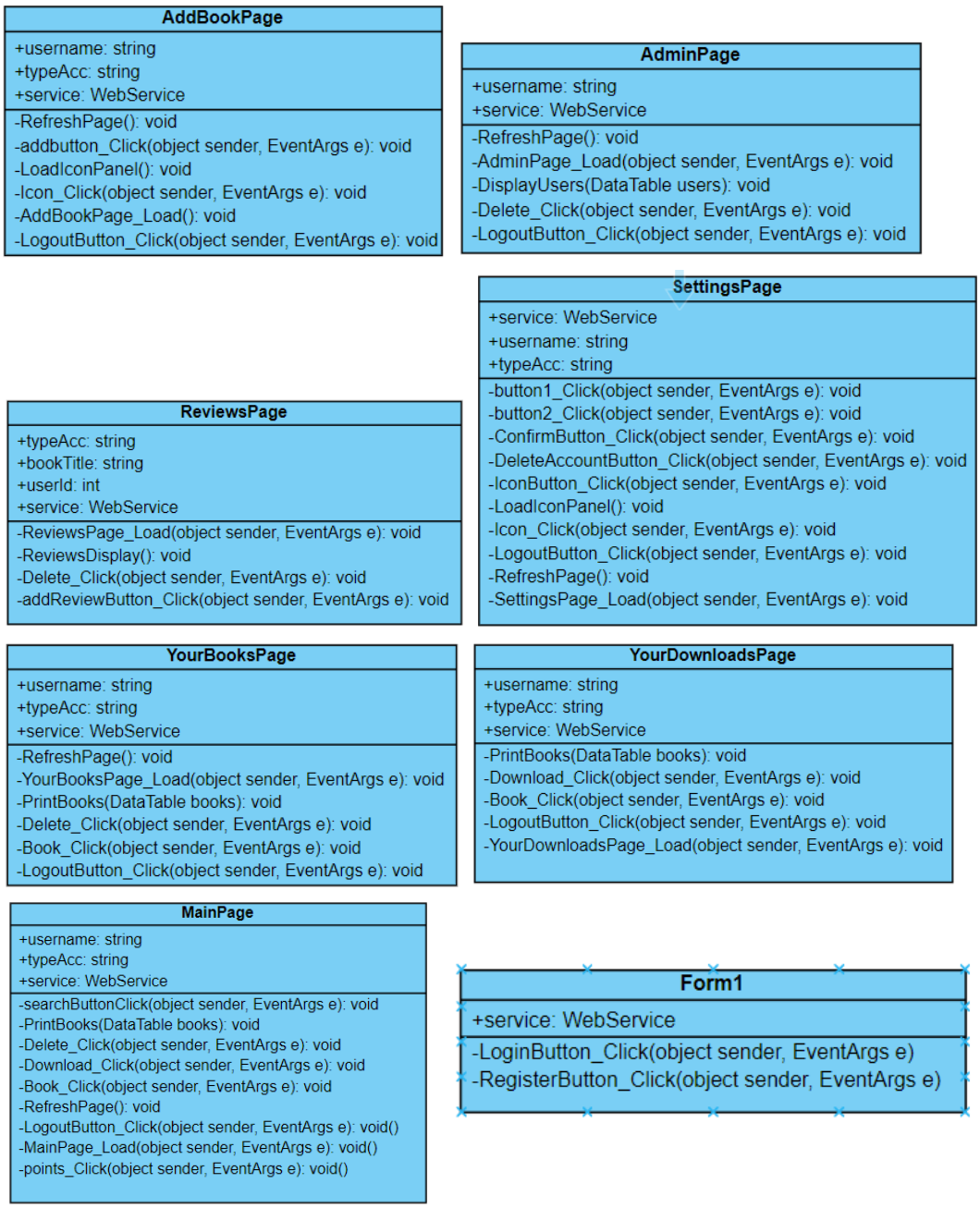
```
)WITH  (PAD_INDEX  =  OFF,  STATISTICS_NORECOMPUTE  =  OFF,  IGNORE_DUP_KEY  =
OFF,      ALLOW_ROW_LOCKS      =      ON,      ALLOW_PAGE_LOCKS      =      ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO

/****** Object:  Table [dbo].[Icons]
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Icons](
    [id] [int] NOT NULL,
    [imgpath] [varchar](max) NOT NULL,
 CONSTRAINT [PK_Icon] PRIMARY KEY CLUSTERED
(
    [id] ASC
)WITH  (PAD_INDEX  =  OFF,  STATISTICS_NORECOMPUTE  =  OFF,  IGNORE_DUP_KEY  =
OFF,      ALLOW_ROW_LOCKS      =      ON,      ALLOW_PAGE_LOCKS      =      ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
/****** Object:  Table [dbo].[Reviews]
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Reviews](
    [id] [int] NOT NULL,
    [bookId] [int] NOT NULL,
    [userId] [int] NOT NULL,
    [text] [varchar](300) NULL,
    [rating] [float] NOT NULL,
 CONSTRAINT [PK_Reviews] PRIMARY KEY CLUSTERED
(
    [id] ASC
)WITH  (PAD_INDEX  =  OFF,  STATISTICS_NORECOMPUTE  =  OFF,  IGNORE_DUP_KEY  =
OFF,      ALLOW_ROW_LOCKS      =      ON,      ALLOW_PAGE_LOCKS      =      ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[Users]
SET ANSI_NULLS ON
```

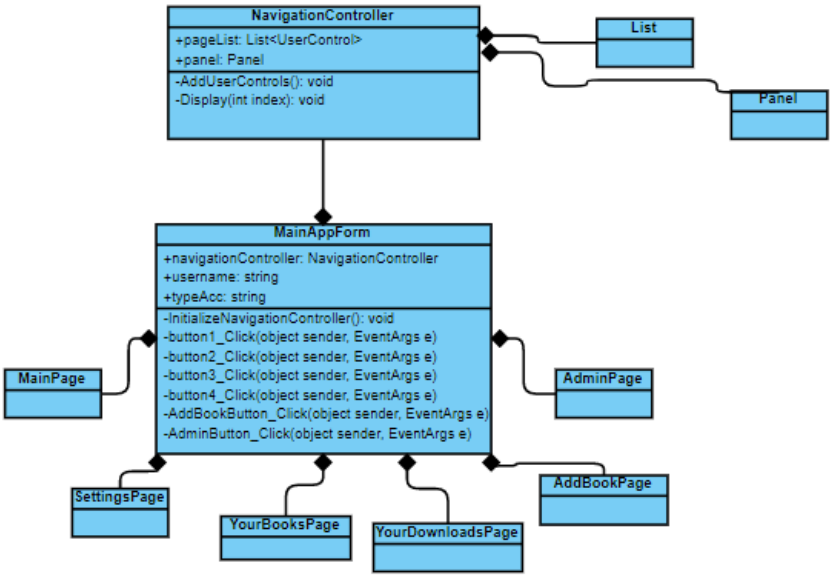
```
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Users](
    [id] [int] NOT NULL,
    [username] [varchar](20) NOT NULL,
    [password] [varchar](255) NULL,
    [points] [int] NULL,
    [typeOfAccount] [varchar](20) NOT NULL,
    [idIcon] [int] NOT NULL,
    [email] [varchar](50) NULL,
 CONSTRAINT [PK_Users] PRIMARY KEY CLUSTERED
(
    [id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Books] WITH CHECK ADD CONSTRAINT [FK_Books_Icons]
FOREIGN KEY([idIconBook])
REFERENCES [dbo].[IconBooks] ([id])
ON UPDATE CASCADE
ON DELETE CASCADE
GO
ALTER TABLE [dbo].[Books] CHECK CONSTRAINT [FK_Books_Icons]
GO
ALTER TABLE [dbo].[Books] WITH CHECK ADD CONSTRAINT [FK_Books_Users]
FOREIGN KEY([idAuthor])
REFERENCES [dbo].[Users] ([id])
ON UPDATE CASCADE
ON DELETE CASCADE
GO
ALTER TABLE [dbo].[Books] CHECK CONSTRAINT [FK_Books_Users]
GO
ALTER TABLE [dbo].[Users] WITH CHECK ADD CONSTRAINT [FK_Users_Icon]
FOREIGN KEY([idIcon])
REFERENCES [dbo].[Icons] ([id])
ON UPDATE CASCADE
ON DELETE CASCADE
GO
ALTER TABLE [dbo].[Users] CHECK CONSTRAINT [FK_Users_Icon]
```

3.3. Application Implementation

3.3.1. Client File Structure

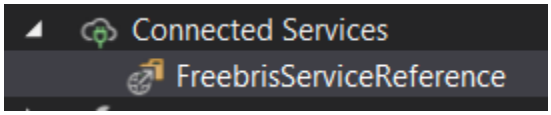
The file structure of the project is provided with the following UML diagrams structures:





The client file structure for the C# the project is organized in a way that promotes modularity, organization, and ease of development.

The first and the most important aspect of our application is establishing a connection to the server through a service reference.



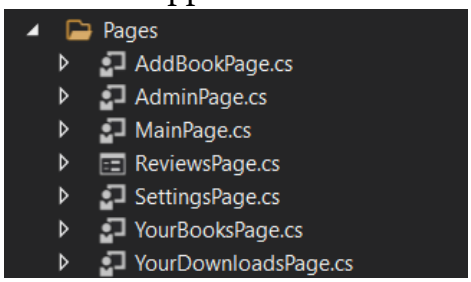
The Login and Register Pages are separated, the method for login being the next one..

```
1 reference
private void loginbtn_Click_1(object sender, EventArgs e)
{
    if (service.CheckPassword(username: usernameLogTextBox.Text, password: PassLogTextBox.Text))
    {
        string typeAcc;
        if (service.IsAdmin(username: usernameLogTextBox.Text))
        {
            typeAcc = "admin";
        }
        else
        {
            typeAcc = "classic";
        }

        MainAppForm homePage = new MainAppForm(username: usernameLogTextBox.Text, typeAcc: typeAcc);
        this.Hide();
        homePage.Show();
        //MessageBox.Show("OK");
    }
    else
    {
        MessageBox.Show(text: "NOT OK");
    }
}
```

And the one for registration...

```
private void registerbtn_Click(object sender, EventArgs e)
{
    if (PassRegTextBox.Text == Pass2RegTextBox.Text && emailRegTextBox.Text == email2RegTextBox.Text)
    {
        if (!service.CreateUser(username: UserRegTextBox.Text, password: PassRegTextBox.Text, email: emailRegTextBox.Text))
        {
            MessageBox.Show(text: "Username already exits");
        }
        else
        {
            MainAppForm homePage = new MainAppForm(username: UserRegTextBox.Text, typeAcc: "classic");
            MessageBox.Show(text: "User created");
            this.Hide();
            homePage.Show();
        }
    }
    else
    {
        if (PassRegTextBox.Text == Pass2RegTextBox.Text)
        {
            MessageBox.Show(text: "Passwords do not match!");
        }
        else
        {
            MessageBox.Show(text: "Emails do not match!");
        }
    }
}
```

We also have a folder containing the “pages” of the application. In our structure, the "Pages" folder contains individual C# files representing different pages of the application. Each page class is responsible for the functionality and presentation of its respective page. This structure allows for easy navigation and management of the pages within the application.



The elements in the "Pages" folder are user control objects that contain the code for every functionality a user can perform on that page.

The **AddBookPage** class starts with a reference to the service and a constructor

```
public partial class AddBookPage : UserControl
{
    string username;
    string typeAcc;
    FreebrisServiceReference.FreebrisWebServiceSoapClient service = new FreebrisServiceReference.FreebrisWebServiceSoapClient();
    1 reference
    public AddBookPage(string username, string typeAcc)
    {
        InitializeComponent();
        textBox1.Text = "";
        textBox2.Text = "";
        this.username = username;
        this.typeAcc = typeAcc;
        points.Text = service.GetPoints(username: username).ToString();
        LoadIconPanel();
    }
```

Then we have the icons load method for interface with a special click event method of each icon.

```
1 reference
private void LoadIconPanel()
{
    panel1.Controls.Clear();
    panel1.AutoScroll = true;
    DataTable icons = service.GetAllBookIcons();
    int y = 0;
    for (int i = 0; i < icons.Rows.Count; i++)
    {
        Button icon = new Button();
        icon.Name = icons.Rows[index: i][columnName: "id"].ToString();
        icon.Size = new Size(width: 100, height: 100);
        string file = icons.Rows[index: i][columnName: "imgpath"].ToString();
        icon.BackgroundImage = Image.FromFile(filename: file);
        icon.BackgroundImageLayout = ImageLayout.Stretch;
        icon.Visible = true;
        icon.FlatStyle = System.Windows.Forms.FlatStyle.Flat;
        icon.Location = new Point(x: 120, y: y);
        icon.Click += new EventHandler(Icon_Click);
        y += 150;

        panel1.Controls.Add(value: icon);
    }
}
1 reference
private void Icon_Click(object sender, EventArgs e)
{
    if (textBox1.Text != "" && textBox2.Text != "")
    {
        DialogResult dialogResult = MessageBox.Show(text: "Do you want to pick this book cover? After you pick the book will be uploaded.", caption: "Book", buttons: MessageBoxButtons.YesNo, icon: MessageBoxIcon.Question);
        if (dialogResult == DialogResult.Yes)
        {
            Button btn = (Button)sender;
            if (btn != null)
            {
                int idUser = service.GetId(username: username);
                service.CreateBook(name: textBox2.Text, size: 0, idAuthor: idUser, idIconBook: int.Parse(s: btn.Name), path: textBox1.Text);
                MessageBox.Show(text: "Your book had been uploaded! ");
                RefreshPage();
            }
        }
    }
    else
    {
        MessageBox.Show(text: "Choose a pdf and a title first!");
    }
}
```

The last important method, and the one that is used on each page, is the logout one. This is an event method that closes the application.

```
1 reference
private void LogoutButton_Click(object sender, EventArgs e)
{

    System.Windows.Forms.Application.Exit();
}
```

The next one is the **AdminPage**. This is a specific page that is only accessible to admins and is used to remove accounts of problematic users.
It starts with the constructor and load methods.

```
string username;
FreebrisServiceReference.FreebrisWebServiceSoapClient service = new FreebrisServiceReference.FreebrisWebServiceSoapClient();

1 reference
public AdminPage(string username)
{
    InitializeComponent();
    this.username = username;
    points.Text = service.GetPoints(username: username).ToString();
}
1 reference
private void AdminPage_Load(object sender, EventArgs e)
{
    DataTable users = service.GetAllUsers();
    DisplayUsers(users: users);
}
```

Next, we have the "Display Users" methods, which generate a visually appealing list of all users, including their icons and names. Additionally, this display includes buttons that allow administrators to easily remove problematically user accounts, along with a delete event method.

```
2 references
private void DisplayUsers(DataTable users)
{
    panel1.Controls.Clear();
    panel1.AutoScroll = true;
    if (users.Rows.Count == 0)
    {
        Label message = new Label();
        message.Text = "No users to be shown!";
        message.Location = new Point(x: 0, y: 100);
        message.Size = new Size(width: 100, height: 30);
        message.Visible = true;

        panel1.Controls.Add(value: message);
        return;
    }

    int y = 0;
    for (int i = 0; i < users.Rows.Count; i++)
    {
        Button book = new Button();
        book.Name = users.Rows[index: i][columnName: "username"].ToString();
        book.Size = new Size(width: 200, height: 200);
        string file = service.GetIconBook(id: int.Parse(s: users.Rows[index: i][columnName: "idIcon"].ToString()));
        book.BackgroundImage = Image.FromFile(filename: file);
        book.BackgroundImageLayout = ImageLayout.Stretch;
        book.Visible = true;
        book.FlatStyle = System.Windows.Forms.FlatStyle.Flat;
        book.Location = new Point(x: 120, y: y);

        Label tb = new Label();
        tb.Text = users.Rows[index: i][columnName: "username"].ToString();
        tb.Location = new Point(x: 350, y: y);
        tb.Visible = true;

        Button delete = new Button();
        delete.Name = users.Rows[index: i][columnName: "id"].ToString();
        delete.Size = new Size(width: 80, height: 40);
        delete.Text = "Remove";
        delete.Visible = true;
        delete.Location = new Point(x: 0, y: y + 20);
        delete.Click += new EventHandler(Delete_Click);

        panel1.Controls.Add(value: book);
        panel1.Controls.Add(value: tb);
        panel1.Controls.Add(value: delete);

        y += 250;
    }
}
```

```
private void Delete_Click(object sender, EventArgs e)
{
    DialogResult dialogResult = MessageBox.Show(text: "Are you sure you want to delete this account?", caption: "Warning",
        buttons: MessageBoxButtons.YesNo, icon: MessageBoxIcon.Warning);
    if (dialogResult == DialogResult.Yes)
    {
        Button btn = (Button)sender;
        service.DeleteUser(id: int.Parse(s: btn.Name));
        MessageBox.Show(text: "The account had been deleted");
        DataTable users = service.GetAllUsers();
        DisplayUsers(users: users);
    }
}
```

The next page in the list is the **MainPage**.

```
string username;
string typeAcc;
FreebrisServiceReference.FreebrisWebServiceSoapClient service = new FreebrisServiceReference.FreebrisWebServiceSoapClient();
1 reference
public MainPage(string username, string typeAcc)
{
    this.username = username;
    this.typeAcc = typeAcc;
    InitializeComponent();
    DataTable books = service.GetAllBooks();
    PrintBooks(books: books);
    points.Text = service.GetPoints(username: username).ToString();
}
```

We have a search button that gives different results based on the checked filter.

```
private void searchButton_Click(object sender, EventArgs e)
{
    // by author
    if(radioButton1.Checked)
    {
        DataTable books = service.GetBooksByAuthor(authorName: textBox1.Text);
        PrintBooks(books: books);
    }
    //by title
    else if (radioButton2.Checked)
    {
        DataTable books = service.GetBooksByTitle(bookName: textBox1.Text);
        PrintBooks(books: books);
    }
    // all
    else
    {
        DataTable books = service.GetAllBooks();
        PrintBooks(books: books);
    }
}
```

The most important method here is the Printing method that displays the books stored in the database in a pleasant way, along with different functionalities that the users can apply on them. This method appears in variations in other pages, listing different functionalities.

```
55      private void PrintBooks(DataTable books)
56      {
57          panel1.Controls.Clear();
58          panel1.AutoScroll = true;
59          int y=0;
60          if(books.Rows.Count == 0)
61          {
62              Label message = new Label();
63              message.Text = "No books to be shown!";
64              message.Location = new Point(x: 0, y: 100);
65              message.Size = new Size(width: 100, height: 30);
66              message.Visible = true;
67
68              panel1.Controls.Add(value: message);
69              return;
70          }
71          for(int i=0; i<books.Rows.Count; i++)
72          {
73              Button book = new Button();
74              book.Name = books.Rows[index: i][columnName: "name"].ToString();
75              book.Size = new Size(width: 200, height: 200);
76              string file = service.GetIconBook(id: int.Parse(s: books.Rows[index: i][columnName: "idIconBook"].ToString()));
77              book.BackgroundImage = Image.FromFile(filename: file);
78              book.BackgroundImageLayout = ImageLayout.Stretch;
79              book.Visible = true;
80              book.FlatStyle = System.Windows.Forms.FlatStyle.Flat;
81              book.Location = new Point(x: 120, y: y);
82              book.Click += new EventHandler(Book_Click);
83
84              Label tb = new Label();
85              tb.Text = books.Rows[index: i][columnName: "name"].ToString();
86              tb.Location = new Point(x: 350, y: y);
87              tb.Visible = true;
88              tb.Size = new Size(width: 200, height: 200);
89
90              Label tb2 = new Label();
91              string author = service.GetUsername(id: int.Parse(s: books.Rows[index: i][columnName: "idAuthor"].ToString()));
92              if(author == null)
93              {
94                  author = "unknown";
95              }
96              tb2.Text = author;
97              tb2.Location = new Point(x: 590, y: y);
```

```
97              tb2.Location = new Point(x: 590, y: y);
98              tb2.Visible = true;
99
100             Button download = new Button();
101             download.Name = books.Rows[index: i][columnName: "name"].ToString();
102             download.Size = new Size(width: 80, height: 40);
103             download.Text = "Get this book";
104             download.Visible = true;
105             download.Location = new Point(x: 0, y: y);
106             download.Click += new EventHandler(Download_Click);
107
108             if (service.IsAdmin(username: username))
109             {
110                 Button delete = new Button();
111                 delete.Name = books.Rows[index: i][columnName: "name"].ToString();
112                 delete.Size = new Size(width: 80, height: 40);
113                 delete.Text = "Delete";
114                 delete.Visible = true;
115                 delete.Location = new Point(x: 0, y: y + 120);
116                 delete.Click += new EventHandler(Delete_Click);
117                 panel1.Controls.Add(value: delete);
118             }
119
120             Label path = new Label();
121             path.Name = books.Rows[index: i][columnName: "name"].ToString() + "label";
122             path.Text = author;
123             path.Location = new Point(x: 620, y: y);
124             path.Visible = false;
125
126             Label points = new Label();
127             points.Text = "Points required: 20";
128             points.Location = new Point(x: 0, y: y+60);
129             points.Visible = true;
130
131             panel1.Controls.Add(value: book);
132             panel1.Controls.Add(value: tb);
133             panel1.Controls.Add(value: tb2);
134             panel1.Controls.Add(value: download);
135             panel1.Controls.Add(value: points);
136             panel1.Controls.Add(value: path);
137
138              y += 250;
139         }
140     }
```

We have events for the buttons..

```
private void Delete_Click(object sender, EventArgs e)
{
    DialogResult dialogResult = MessageBox.Show(text: "Are you sure you want to delete your book?", caption: "Warning",
        buttons: MessageBoxButtons.YesNo, icon: MessageBoxIcon.Warning);
    if (dialogResult == DialogResult.Yes)
    {
        Button btn = (Button)sender;
        int id = service.GetBookId(bookTitle: btn.Name);
        service.DeleteBook(id: id);
        MessageBox.Show(text: "Your book had been deleted");

        DataTable books = service.GetBooksByAuthor(authorName: username);
        RefreshPage();
    }
}
1 reference
private void Book_Click(object sender, EventArgs e)
{
    Button btn = (Button)sender;
    int userId = service.GetId(username: username);
    ReviewsPage reviewsPage = new ReviewsPage(bookTitle: btn.Name, userId: userId, typeAcc: typeAcc);
    reviewsPage.ShowDialog();
}
1 reference
private void Download_Click(object sender, EventArgs e)
{
    int idUser = service.GetId(username: username);
    Button btn = (Button)sender;
    int idBook = service.GetBookId(bookTitle: btn.Name);

    if (service.GetPoints(username: username) < 20)
    {
        MessageBox.Show(text: "You don't have enough points to download this book! ");
        return;
    }
    string email = service.GetEmail(username: username);
    service.CreateDownload(idUser: idUser, idBook: idBook);
    string location = service.GetPath(bookId: idBook);
    service.SendEmail(email: email, subject: "Your new book", text: location);
    service.AddPoints(id: idUser, points: -20);
    points.Text = service.GetPoints(username: username).ToString();

    MessageBox.Show(text: "The book was sent on email!");
}
```

And a refresh page method

```
private void RefreshPage()
{
    this.Show();
    this.Controls.Clear();
    InitializeComponent();
    points.Text = service.GetPoints(username: username).ToString();
    DataTable books = service.GetAllBooks();
    PrintBooks(books: books);
}
```

The **ReviewsPage** is a form, appearing as a pop-up when clicking a book.

```
public partial class ReviewsPage : Form
{
    string typeAcc;
    string bookTitle;
    int userId;
    FreebrisServiceReference.FreebrisWebServiceSoapClient service = new FreebrisServiceReference.FreebrisWebServiceSoapClient();
    3 references
    public ReviewsPage(string bookTitle, int userId, string typeAcc)
    {
        this.typeAcc = typeAcc;
        this.bookTitle = bookTitle;
        this.userId = userId;
        InitializeComponent();
    }

    1 reference
    private void ReviewsPage_Load(object sender, EventArgs e)
    {
        panel1.Controls.Clear();
        panel1.AutoScroll = true;
        ReviewsDisplay();
    }
```

The most important method on the page is the generate-display one

```
private void ReviewsDisplay()
{
    panel1.Controls.Clear();
    panel1.AutoScroll = true;

    DataTable reviews = service.GetAllReviewForBook(bookTitle: bookTitle);

    int y = 0;
    for(int i=0;i<reviews.Rows.Count;i++)
    {
        Label user = new Label();
        string username = service.GetUsername(id: int.Parse(s: reviews.Rows[index: i][columnName: "userId"].ToString()));
        user.Text = username;
        user.Location = new Point(x: 0, y: y+20);
        user.Visible = true;

        Label reviewText = new Label();
        reviewText.Text = reviews.Rows[index: i][columnName: "text"].ToString();
        reviewText.Location = new Point(x: 150, y: y);
        reviewText.Visible = true;
        reviewText.Size = new Size(width: 400, height: 100);
        reviewText.BackColor = Color.White;

        if (service.IsAdmin(username: username))
        {
            Button delete = new Button();
            delete.Name = reviews.Rows[index: i][columnName: "id"].ToString();
            delete.Size = new Size(width: 80, height: 40);
            delete.Text = "Delete";
            delete.Visible = true;
            delete.Location = new Point(x: 0, y: y + 60);
            delete.Click += new EventHandler(Delete_Click);
            panel1.Controls.Add(value: delete);
        }
        else if (int.Parse(s: reviews.Rows[index: i][columnName: "userId"].ToString()) == userId)
        {
            Button delete = new Button();
            delete.Name = reviews.Rows[index: i][columnName: "id"].ToString();
            delete.Size = new Size(width: 80, height: 40);
            delete.Text = "delete";
            delete.Visible = true;
            delete.Location = new Point(x: 600, y: y );
            delete.Click += new EventHandler(Delete_Click);
            panel1.Controls.Add(value: delete);
        }

        panel1.Controls.Add(value: user);
        panel1.Controls.Add(value: reviewText);

        y = y + 120;

    }
}
```

And this comes along with a delete event.

```
2 references
private void Delete_Click(object sender, EventArgs e)
{

    Button btn = (Button)sender;
    service.DeleteReviewById(id: int.Parse(s: btn.Name));
    ReviewsDisplay();
}
```

The next one is the **SettingsPage.** This one has buttons that allows the user to select the interface for the action they need to perform.

```
// change password
1 reference
private void button1_Click(object sender, EventArgs e)
{
    tableLayoutPanelChangePass.Show();
    tableLayoutPanelChangeEmail.Hide();
    IconPanel.Visible = false;
    ConfirmButton.Show();

}
// change email
1 reference
private void button2_Click(object sender, EventArgs e)
{
    tableLayoutPanelChangePass.Hide();
    tableLayoutPanelChangeEmail.Show();
    IconPanel.Visible = false;
    ConfirmButton.Show();
}
```

```
private void IconButton_Click(object sender, EventArgs e)
{
    tableLayoutPanelChangeEmail.Visible = false;
    tableLayoutPanelChangePass.Visible = false;
    IconPanel.Visible = true;
    ConfirmButton.Visible = false;
    LoadIconPanel();
}
```

And there is the Confirm button that actually performs the action.

```
1 reference
private void ConfirmButton_Click(object sender, EventArgs e)
{
    if(tableLayoutPanelChangeEmail.Visible == true)
    {

    }
    else if(tableLayoutPanelChangePass.Visible == true)
    {
        if(textBox2.Text == textBox3.Text)
        {
            if(textBox1.Text != textBox3.Text)
            {
                service.ChangePassword(username: username, password: textBox3.Text);
                MessageBox.Show(text: "The password is changed!");
            }
            else
            {
                MessageBox.Show(text: "The new password is the same as the old one!");
            }
        }
        else
        {
            MessageBox.Show(text: "The passwords are not the same!");
        }
    }
    else if(IconPanel.Visible == true)
    {

    }
}
```

We have the delete account action.

```
private void DeleteAccountButton_Click(object sender, EventArgs e)
{
    DialogResult dialogResult = MessageBox.Show(text: "Are you sure you want to delete your account?", caption: "Warning",
        buttons: MessageBoxButtons.YesNo, icon: MessageBoxIcon.Warning);
    if (dialogResult == DialogResult.Yes)
    {
        int id = service.GetId(username: username);
        service.DeleteUser(id: id);
        MessageBox.Show(text: "Your account had been deleted");
        // Logout!!!!!!!!!!
    }
}
```

The method that loads the icons, so that the user can pick one.

```
private void LoadIconPanel()
{
    IconPanel.Controls.Clear();
    DataTable icons = service.GetAllIcons();
    int y = 0;
    for(int i= 0;i < icons.Rows.Count;i++)
    {
        Button icon = new Button();
        icon.Name = icons.Rows[index: i][columnName: "id"].ToString();
        icon.Size = new Size(width: 100, height: 100);
        string file = icons.Rows[index: i][columnName: "imgpath"].ToString();
        icon.BackgroundImage = Image.FromFile(filename: file);
        icon.BackgroundImageLayout = ImageLayout.Stretch;
        icon.Visible = true;
        icon.FlatStyle = System.Windows.Forms.FlatStyle.Flat;
        icon.Location = new Point(x: 120, y: y);
        icon.Click += new EventHandler(Icon_Click);
        y += 150;

        IconPanel.Controls.Add(value: icon);
    }
}
```

```
private void Icon_Click(object sender, EventArgs e)
{
    DialogResult dialogResult = MessageBox.Show(text: "Do you want to pick this profile icon?", caption: "Icon",
        buttons: MessageBoxButtons.YesNo, icon: MessageBoxIcon.Question);
    if (dialogResult == DialogResult.Yes)
    {
        Button btn = (Button)sender;
        if (btn != null)
        {
            int id = service.GetId(username: username);
            service.SetIcon(id: id, idIcon: int.Parse(s: btn.Name));
            MessageBox.Show(text: "Your icon had been set! ");
            RefreshPage();
        }
    }
}
```

The **YourBookPage** displays the books written or uploaded by the user. It contains a similar method as Main page to display the books, with similar functionalities.

The next one is the **YourDownloadPage.** This allows the user to see the book he/she received on email and get them again. The main, generating method is similar to the one form the main page.

All these pages are displayed in a panel. in a **MainAppForm**. This contains a navigation controller object that allows the pages to swap.

```
public partial class MainAppForm : Form
{
    NavigationController navigationController;
    string username;
    string typeAcc;
    3 references
    public MainAppForm(string username, string typeAcc)
    {
        this.username = username;
        this.typeAcc = typeAcc;
        InitializeComponent();
        InitializeNavigationController();
        if(typeAcc != "admin")
        {
            AdminButton.Visible = false;
        }
    }

    1 reference
    private void InitializeNavigationController()
    {
        List<UserControl> pages = new List<UserControl>()
        {
            new MainPage(username: username, typeAcc: typeAcc),
            new SettingsPage(username: username, typeAcc: typeAcc),
            new YourBooksPage(username: username, typeAcc: typeAcc),
            new YourDownloadsPage(username: username, typeAcc: typeAcc),
            new AddBookPage(username: username, typeAcc: typeAcc),
            new AdminPage(username: username)

            // more pages to add
        };

        navigationController = new NavigationController(pageList: pages, panel: panel1);
        navigationController.Display(index: 0);
    }
```

And the event methods for each button to display the right page.

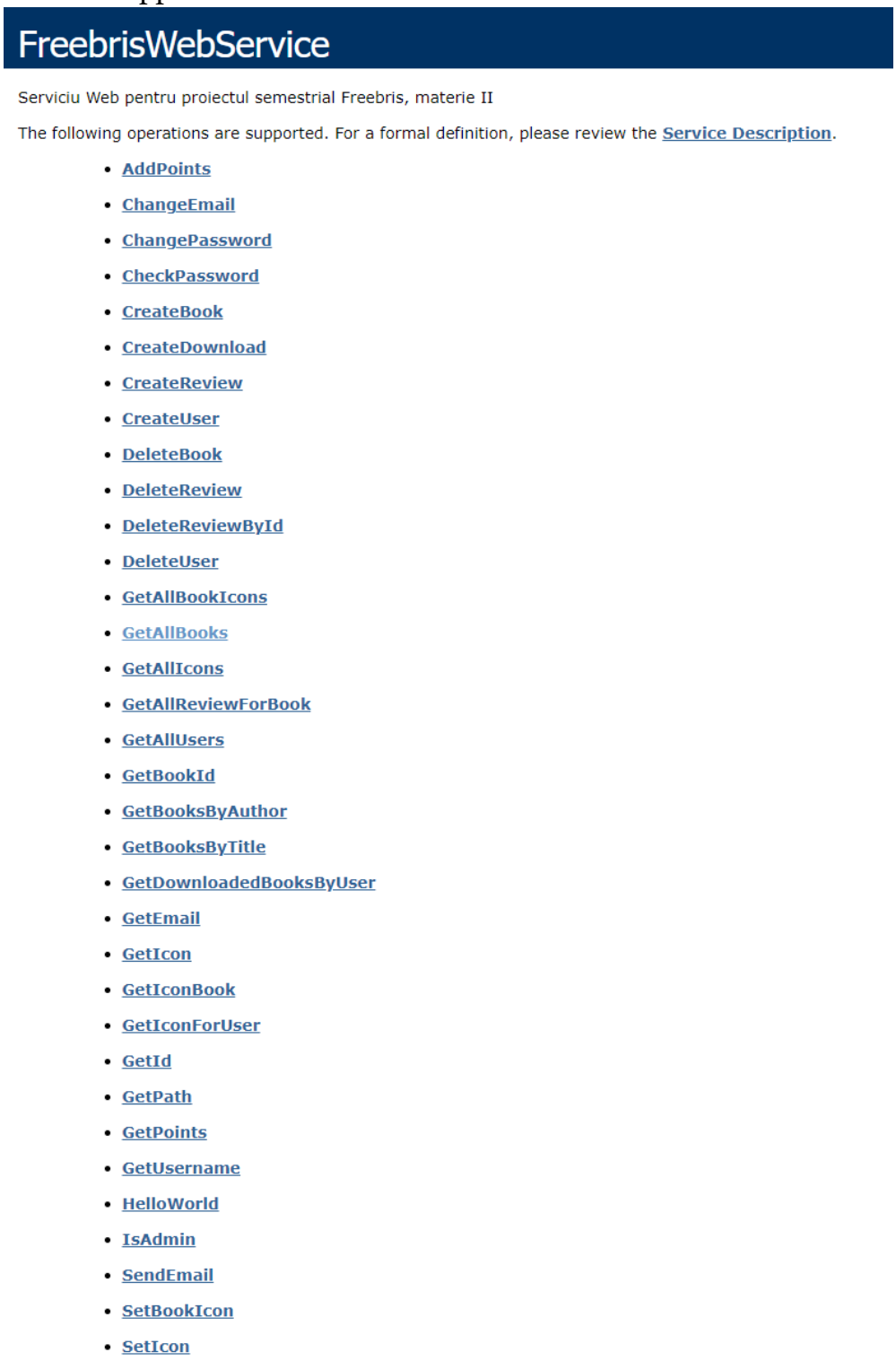
```
private void button1_Click(object sender, EventArgs e)
{
    navigationController.Display(index: 0);
}

1 reference
private void button2_Click(object sender, EventArgs e)
{
    navigationController.Display(index: 1);
}

1 reference
private void button3_Click(object sender, EventArgs e)
{
    navigationController.Display(index: 2);
}

1 reference
private void button4_Click(object sender, EventArgs e)
{
    navigationController.Display(index: 3);
}

1 reference
private void AddBookButton_Click(object sender, EventArgs e)
{
    navigationController.Display(index: 4);
}

1 reference
private void AdminButton_Click(object sender, EventArgs e)
{
    navigationController.Display(index: 5);
}
```

The **NavigationController** is a class that has the purpose of storing a list of the pages and making the application as fluid as possible. It also has a panel in which it has to display the pages.

```
class NavigationController
{
    List<UserControl> pageList = new List<UserControl>();
    Panel panel;

    1 reference
    public NavigationController(List<UserControl> pageList, Panel panel)
    {
        this.pageList = pageList;
        this.panel = panel;
        AddUserControls();
    }

    1 reference
    private void AddUserControls()
    {
        for (int i = 0; i < pageList.Count(); i++)
        {
            pageList[index: i].Dock = DockStyle.Fill;
            panel.Controls.Add(value: pageList[index: i]);
        }
    }

    7 references
    public void Display(int index)
    {
        if (index < pageList.Count())
        {
            pageList[index: index].BringToFront();
        }
    }

}
```

3.3.2. Server File Structure

The server file structure of a web service plays a crucial role in organizing and maintaining the codebase. On the application server, which runs on https://localhost:44348/FreebrisWebService.asmx, there are several methods available for client invocation. These methods serve as the server-side functionality accessible to the client application.



This part provides an overview of the server file structure for our web service that utilizes CRUD (Create, Read, Update, Delete) and other operations implemented in C#.

-CREATE - used to add a new User to the database

```
1 reference
public void AddUserToDB(int id, string username, string password, string email)
{
    SqlConnection connection = new SqlConnection();
    SqlCommand cmd = new SqlCommand("INSERT INTO Users VALUES ('" + id + "', '" + username + "', '" + password + "', '" + 100 + "', '" + "classic', '"
    connection.ConnectionString = ConfigurationManager.ConnectionStrings["ConnectionWebService"].ToString();
    connection.Open();
    cmd.ExecuteReader();
}

[WebMethod]
0 references
public bool CreateUser(string username, string password, string email)
{
    SqlConnection connection = new SqlConnection();
    SqlCommand cmd = new SqlCommand("SELECT username FROM Users WHERE username = \'" + username + "\'", connection);
    connection.ConnectionString = ConfigurationManager.ConnectionStrings["ConnectionWebService"].ToString();
    connection.Open();
    SqlDataReader dr = cmd.ExecuteReader();
    byte[] hashedPassword;
    if (dr.HasRows)
    {
        return false;
    }
    else
    {
        int id = GenerateId("Users");
        var sha = SHA256.Create();
        var asByteArray = Encoding.Default.GetBytes(password);
        hashedPassword = sha.ComputeHash(asByteArray);
        AddUserToDB(id, username, Convert.ToBase64String(hashedPassword), email);
    }
    return true;
}
```

-CREATE used to add a new Review to the database by giving the bookId to which the review is for, the userId of the user that leaves the review and the actual review.

```
[WebMethod]
0 references
public bool CreateReview(int bookId, int userId, string text)
{
    try
    {
        int id = GenerateId("Reviews");
        SqlConnection connection = new SqlConnection();
        SqlCommand cmd = new SqlCommand("INSERT INTO Reviews VALUES ('" + id + "', '" + bookId + "', '" + userId + "', '" + text + "', '" + "1" + "')", connection);
        connection.ConnectionString = ConfigurationManager.ConnectionStrings["ConnectionWebService"].ToString();
        connection.Open();
        cmd.ExecuteReader();
    }
    catch (Exception ex)
    {
        return false;
    }
    AddPoints(userId, 5);
    return true;
}
```

-CREATE used to add a new Download to the database which has all the books that have been downloaded with all the users that downloaded.

```
[WebMethod]
0 references
public void CreateDownload(int idUser, int idBook)
{
    int id = GenerateId("Downloads");
    SqlConnection connection = new SqlConnection();
    SqlCommand cmd = new SqlCommand("INSERT INTO Downloads VALUES ('" + id + "', '" + idUser + "', '" + idBook + "')", connection);
    connection.ConnectionString = ConfigurationManager.ConnectionStrings["ConnectionWebService"].ToString();
    connection.Open();
    cmd.ExecuteReader();
}
```

-CREATE used to add a new Book to the database with its author and the actual book

```
[WebMethod]
0 references
public void CreateBook(string name, int size, int idAuthor, int idIconBook, string path)
{
    int id = GenerateId("Books");
    SqlConnection connection = new SqlConnection();
    SqlCommand cmd = new SqlCommand("INSERT INTO Books VALUES ('" + id + "', '" + name + "', '" + size + "', '" + path + "', '" + idAuthor + "', '" + idIconBook + "')", connection);
    connection.ConnectionString = ConfigurationManager.ConnectionStrings["ConnectionWebService"].ToString();
    connection.Open();
    cmd.ExecuteReader();

}
```

-GET used to receive a DataTable with all the users from the db

```
[WebMethod]
0 references
public DataTable GetAllUsers()
{
    SqlConnection connection = new SqlConnection();
    SqlCommand cmd = new SqlCommand("SELECT * FROM Users", connection);
    connection.ConnectionString = ConfigurationManager.ConnectionStrings["ConnectionWebService"].ToString();
    connection.Open();
    SqlDataReader dr = cmd.ExecuteReader();

    DataTable dt = new DataTable();
    dt.Load(dr);
    dt.TableName = "Users";
    return dt;
}
```

-GET used to receive a DataTable with all the icons from the db

```
[WebMethod]
0 references
public DataTable GetAllIcons()
{
    SqlConnection connection = new SqlConnection();
    SqlCommand cmd = new SqlCommand("SELECT * FROM Icons", connection);
    connection.ConnectionString = ConfigurationManager.ConnectionStrings["ConnectionWebService"].ToString();
    connection.Open();
    SqlDataReader dr = cmd.ExecuteReader();

    DataTable dt = new DataTable();
    dt.Load(dr);
    dt.TableName = "Icons";
    return dt;
}
```

-GET used to receive a DataTable with all the reviews that a book

```
[WebMethod]
0 references
public DataTable GetAllReviewForBook(string bookTitle)
{
    SqlConnection connection = new SqlConnection();
    int bookId = GetBookId(bookTitle);
    SqlCommand cmd = new SqlCommand("SELECT * FROM Reviews WHERE bookId = \'" + bookId + "\'", connection);
    connection.ConnectionString = ConfigurationManager.ConnectionStrings["ConnectionWebService"].ToString();
    connection.Open();
    SqlDataReader dr = cmd.ExecuteReader();

    DataTable dt = new DataTable();
    dt.Load(dr);
    dt.TableName = "Reviews";
    return dt;
}
```

-GET used to receive the id of a book by title

```
[WebMethod]
1 reference
public int GetBookId(string bookTitle)
{
    SqlConnection connection = new SqlConnection();
    SqlCommand cmd = new SqlCommand("SELECT id FROM Books WHERE name = '" + bookTitle + "'", connection);
    connection.ConnectionString = ConfigurationManager.ConnectionStrings["ConnectionWebService"].ToString();
    connection.Open();
    object result = cmd.ExecuteScalar();

    if (result != null)
    {
        return Convert.ToInt32(result);
    }
    else
    {
        return -1;
    }
}
```

-GET used to receive the path of a book by its id

```
[WebMethod]
0 references
public string GetPath(int bookId)
{
    SqlConnection connection = new SqlConnection();
    SqlCommand cmd = new SqlCommand("SELECT pdfFile FROM Books WHERE id = \'" + bookId + "\'", connection);
    connection.ConnectionString = ConfigurationManager.ConnectionStrings["ConnectionWebService"].ToString();
    connection.Open();
    SqlDataReader dr = cmd.ExecuteReader();
    if (dr.HasRows)
    {
        while (dr.Read())
        {
            return dr.GetString(0);
        }
    }
    return null;
}
```

-GET used to receive the id of an user by its username

```
[WebMethod]
0 references
public int GetId(string username)
{
    SqlConnection connection = new SqlConnection();
    SqlCommand cmd = new SqlCommand("SELECT id FROM Users WHERE username = \'" + username + "\'", connection);
    connection.ConnectionString = ConfigurationManager.ConnectionStrings["ConnectionWebService"].ToString();
    connection.Open();
    int idNr = 0;
    SqlDataReader dr = cmd.ExecuteReader();
    if (dr.HasRows)
    {
        while (dr.Read())
        {
            idNr = dr.GetInt32(0);
        }
    }
    return idNr;
}
```

-GET used to receive the points of an user

```
[WebMethod]
0 references
public int GetPoints(string username)
{
    SqlConnection connection = new SqlConnection();
    SqlCommand cmd = new SqlCommand("SELECT points FROM Users WHERE username = \'" + username + "\'", connection);
    connection.ConnectionString = ConfigurationManager.ConnectionStrings["ConnectionWebService"].ToString();
    connection.Open();
    int idNr = 0;
    SqlDataReader dr = cmd.ExecuteReader();
    if (dr.HasRows)
    {
        while (dr.Read())
        {
            idNr = dr.GetInt32(0);
        }
    }
    return idNr;
}
```

-GET used to receive all the books from the DB

```
[WebMethod]
1 reference
public DataTable GetAllBooks()
{
    SqlConnection connection = new SqlConnection();
    SqlCommand cmd = new SqlCommand("SELECT * FROM Books", connection);
    connection.ConnectionString = ConfigurationManager.ConnectionStrings["ConnectionWebService"].ToString();
    connection.Open();
    SqlDataReader dr = cmd.ExecuteReader();

    DataTable dt = new DataTable();
    dt.Load(dr);
    dt.TableName = "Books";
    return dt;
}
```

-GET used to receive the email of a user by providing its username

```
[WebMethod]
0 references
public string GetEmail(string username)
{
    SqlConnection connection = new SqlConnection();
    SqlCommand cmd = new SqlCommand("SELECT email FROM Users WHERE username = '" + username + "'", connection);
    connection.ConnectionString = ConfigurationManager.ConnectionStrings["ConnectionWebService"].ToString();
    connection.Open();
    SqlDataReader dr = cmd.ExecuteReader();

    while (dr.Read())
    {
        result = result + dr[0].ToString();
    }
    return result;
}
```

-GET used to receive the icon by its id

```
[WebMethod]
0 references
public string GetIcon(int id)
{
    SqlConnection connection = new SqlConnection();
    SqlCommand cmd = new SqlCommand("SELECT imgpath FROM Icons WHERE id = '" + id + "'", connection);
    connection.ConnectionString = ConfigurationManager.ConnectionStrings["ConnectionWebService"].ToString();
    connection.Open();
    object result = cmd.ExecuteScalar();

    return Convert.ToString(result);
}
```

-GET used to receive the username of an user by its id

```
[WebMethod]
0 references
public string GetUsername(int id)
{
    SqlConnection connection = new SqlConnection();
    SqlCommand cmd = new SqlCommand("SELECT username FROM Users WHERE id = '" + id + "'", connection);
    connection.ConnectionString = ConfigurationManager.ConnectionStrings["ConnectionWebService"].ToString();
    connection.Open();
    object result = cmd.ExecuteScalar();

    return Convert.ToString(result);
}
```

-GET used to receive all the downloaded book of an user by its id

```
[WebMethod]
0 references
public DataTable GetDownloadedBooksByUser(int id)
{
    SqlConnection connection = new SqlConnection();
    SqlCommand cmd = new SqlCommand("SELECT * FROM Books INNER JOIN Downloads ON Books.id = Downloads.idBook WHERE idUser = '" + id + "'", connection);
    connection.ConnectionString = ConfigurationManager.ConnectionStrings["ConnectionWebService"].ToString();
    connection.Open();
    SqlDataReader dr = cmd.ExecuteReader();

    DataTable dt = new DataTable();
    dt.Load(dr);
    dt.TableName = "Downloads";
    return dt;
}
```

-UPDATE -We provide the username and the new email. With the username we get the user from the db and we update the old email with the new one provided.

```
[WebMethod]
0 references
public void ChangeEmail(string username, string newEmail)
{
    SqlConnection connection = new SqlConnection();
    SqlCommand cmd = new SqlCommand("UPDATE Users SET email = '" + newEmail + "' WHERE name = '" + username + "'", connection);
    connection.ConnectionString = ConfigurationManager.ConnectionStrings["ConnectionWebService"].ToString();
    connection.Open();
    cmd.ExecuteReader();
}
```

-UPDATE used to update the password of your user. It gets the username to which the password should be changed, then the new password is encrypted and added to the db

```
[WebMethod]
0 references
public bool ChangePassword(string username, string password)
{
    SqlConnection connection = new SqlConnection();

    SqlCommand cmd = new SqlCommand("update Users set password=@password WHERE username = \'" + username + "\'", connection);

    connection.ConnectionString = ConfigurationManager.ConnectionStrings["ConnectionWebService"].ToString();
    connection.Open();

    var sha = SHA256.Create();
    var asByteArray = Encoding.Default.GetBytes(password);
    var hashedPassword = sha.ComputeHash(asByteArray);
    cmd.Parameters.AddWithValue("@password", Convert.ToBase64String(hashedPassword));

    try
    {
        cmd.ExecuteNonQuery();
    }
    catch
    {
        connection.Close();
        return false;
    }
    connection.Close();
    return true;
}
```

-UPDATE used to set a new icon to a username by checking in the db by that username and if its id is find we replace the old icon with the new one

```
public void SetIcon(int id, int idIcon)
{
    SqlConnection connection = new SqlConnection();
    SqlCommand cmd = new SqlCommand("UPDATE Users SET idIcon = '" + idIcon + "' WHERE id = '" + id + "'", connection);
    connection.ConnectionString = ConfigurationManager.ConnectionStrings["ConnectionWebService"].ToString();
    connection.Open();
    cmd.ExecuteReader();
}
```

-UPDATE used to set a new icon to a book by checking in the db by that book and if its id is find we replace the old icon with the new one

```
[WebMethod]
0 references
public void SetBookIcon(int id, int idIcon)
{
    SqlConnection connection = new SqlConnection();
    SqlCommand cmd = new SqlCommand("UPDATE Books SET idIcon = '" + idIcon + "' WHERE id = '" + id + "'", connection);
    connection.ConnectionString = ConfigurationManager.ConnectionStrings["ConnectionWebService"].ToString();
    connection.Open();
    cmd.ExecuteReader();
}
```

DELETE - an id is provided and the method searches through the db if that id exists, if so it is removed from the db.

```
[WebMethod]
0 references
public bool DeleteUser(int id)
{
    SqlConnection connection = new SqlConnection();
    SqlCommand cmd = new SqlCommand("DELETE FROM Users WHERE id = \'" + id + "\'", connection);
    connection.ConnectionString = ConfigurationManager.ConnectionStrings["ConnectionWebService"].ToString();
    connection.Open();
    int cd = cmd.ExecuteNonQuery();
    if (cd == 0)
    {
        return false;
    }
    return true;
}
```

DELETE - an id is provided and the method searches through the db if that id exists, if so it is removed from the db.

```
[WebMethod]
0 references
public bool DeleteBook(int id)
{
    SqlConnection connection = new SqlConnection();
    SqlCommand cmd = new SqlCommand("DELETE FROM Books WHERE id = \'" + id + "\'", connection);
    connection.ConnectionString = ConfigurationManager.ConnectionStrings["ConnectionWebService"].ToString();
    connection.Open();
    int cd = cmd.ExecuteNonQuery();
    if (cd == 0)
    {
        return false;
    }
    return true;
}
```

DELETE - we get the bookId to which the review if for, the userId that created the review and the actual review and it is removed from the db.(used by admin to delete other user review)

```
[WebMethod]
0 references
public bool DeleteReview(int bookId, int userId, string text)
{
    SqlConnection connection = new SqlConnection();
    SqlCommand cmd = new SqlCommand("DELETE FROM Reviews WHERE bookId = \'" + bookId + "\' AND userId = \'" + userId + "\' AND text = \'" + text + "\'", connection);
    connection.ConnectionString = ConfigurationManager.ConnectionStrings["ConnectionWebService"].ToString();
    connection.Open();
    int cd = cmd.ExecuteNonQuery();
    if (cd == 0)
    {
        return false;
    }
    return true;
}
```

DELETE - we get the id of a review and if it exists it is removed from the db.(used by user to delete his review)

```
[WebMethod]
0 references
public bool DeleteReviewById(int id)
{
    SqlConnection connection = new SqlConnection();
    SqlCommand cmd = new SqlCommand("DELETE FROM Reviews WHERE id = \'" + id + "\'", connection);
    connection.ConnectionString = ConfigurationManager.ConnectionStrings["ConnectionWebService"].ToString();
    connection.Open();
    int cd = cmd.ExecuteNonQuery();
    if (cd == 0)
    {
        return false;
    }
    return true;
}
```

FUNCTIONALITATE- the user and the new password is provided, we check if the user exists in the db, if so we encrypt the password provided and compare it to the password form the db and return true if they are equal

```
[WebMethod]
0 references
public bool CheckPassword(string username, string password)
{
    SqlConnection connection = new SqlConnection();
    SqlCommand cmd = new SqlCommand("SELECT password FROM Users WHERE username = \'" + username + "\'", connection);
    connection.ConnectionString = ConfigurationManager.ConnectionStrings["ConnectionWebService"].ToString();
    connection.Open();
    SqlDataReader dr = cmd.ExecuteReader();
    byte[] hashedPassword = { };

    string pass = "";
    if (dr.HasRows)
    {
        while (dr.Read())
        {
            pass = dr.GetString(0);
            pass = pass.Trim();
            var sha = SHA256.Create();
            var asByteArray = Encoding.Default.GetBytes(password);
            hashedPassword = sha.ComputeHash(asByteArray);
        }
    }

    if (pass.Equals(Convert.ToBase64String(hashedPassword)))
    {
        return true;
    }
    return false;
}
```

FUNCTIONALITATE- method that returns true/false if the provided user is admin or classic user by checking its rank from the db.

```
[WebMethod]
0 references
public bool IsAdmin(string username)
{
    SqlConnection connection = new SqlConnection();
    SqlCommand cmd = new SqlCommand("SELECT typeOfAccount FROM Users WHERE username = \'" + username + "\'", connection);
    connection.ConnectionString = ConfigurationManager.ConnectionStrings["ConnectionWebService"].ToString();
    connection.Open();
    SqlDataReader dr = cmd.ExecuteReader();

    string typeOfAccount = "";
    if (dr.HasRows)
    {
        while (dr.Read())
        {
            typeOfAccount = dr.GetString(0);
        }
    }

    if (typeOfAccount == "admin")
    {
        return true;
    }
    return false;
}
```

FUNCTIONALITATE-As the title states this method is used to add points to an user. We search for the userId provided, get the current number of points and add the new ones to it.

```
[WebMethod]
1 reference
public void AddPoints(int id, int points)
{
    SqlConnection connection = new SqlConnection();
    SqlCommand OldPoints = new SqlCommand("SELECT points From Users WHERE id = '" + id + "'", connection);
    connection.ConnectionString = ConfigurationManager.ConnectionStrings["ConnectionWebService"].ToString();
    connection.Open();

    SqlDataReader dr = OldPoints.ExecuteReader();
    int oldPts = 0;
    if (dr.HasRows)
    {
        while (dr.Read())
        {
            oldPts = dr.GetInt32(0);
        }
    }
    oldPts += points;
    SqlConnection connection2 = new SqlConnection();
    connection2.ConnectionString = ConfigurationManager.ConnectionStrings["ConnectionWebService"].ToString();
    connection2.Open();
    SqlCommand refresh = new SqlCommand("UPDATE Users SET points = '" + oldPts + "' WHERE id = '" + id + "'", connection2);
    try
    {
        refresh.ExecuteNonQuery();
    }
    catch(Exception ex)
    {
        connection.Close();
        connection2.Close();
    }
    connection.Close();
    connection2.Close();
}
```

FUNCTIONALITATE-used to send an email with the selected book to the user's provided email. We create a new variable of type MailMessage and add step by step each field provided. Lastly, we create a SmtpClient which allows sending an email using a Simple mail Transfer Protocol server

```
[WebMethod]
0 references
public void SendEmail(string email, string subject, string text)
{
    string fromMail = "ahs.sarah.2002@gmail.com";
    string fromPassword = "hjbxeikvbuxbdfpd";

    MailMessage message = new MailMessage();
    message.From = new MailAddress(fromMail);
    message.Subject = subject;
    message.To.Add(new MailAddress(email));
    message.Body = "Here is your new book. Enjoy!";
    message.Attachments.Add(new Attachment(text));
    message.IsBodyHtml = false;

    var smtpClient = new SmtpClient("smtp.gmail.com")
    {
        Port = 587,
        Credentials = new NetworkCredential(fromMail, fromPassword),
        EnableSsl = true,
    };

    smtpClient.Send(message);
}
```

FUNCTIONALITATE-used in the main search to get a DataTable with all the books sorted either by title or by author. it searched throw the db and returns all the entities that match the requirements

```
[WebMethod]
0 references
public DataTable GetBooksByTitle(string bookName)
{
    DataTable books = GetAllBooks();
    var filteredRows = books.AsEnumerable()
        .Where(row => row.Field<string>("name").IndexOf(bookName, StringComparison.OrdinalIgnoreCase) >= 0)
        .ToList();

    DataTable dt = books.Clone();
    dt.Rows.Clear();
    foreach(DataRow d in filteredRows)
    {
        dt.Rows.Add(d.ItemArray);
    }
    dt.TableName = "Books";
    return dt;
}

[WebMethod]
0 references
public DataTable GetBooksByAuthor(string authorName)
{
    var authorId = GetUserId(authorName);
    SqlConnection connection = new SqlConnection();
    SqlCommand cmd = new SqlCommand("SELECT * FROM Books WHERE idAuthor = '" + authorId + "'", connection);
    connection.ConnectionString = ConfigurationManager.ConnectionStrings["ConnectionWebService"].ToString();
    connection.Open();

    SqlDataReader dr = cmd.ExecuteReader();
    DataTable dt = new DataTable();
    dt.Load(dr);
    dt.TableName = "Books";
    return dt;
}
```

FUNCTIONALITATE- used to create ids for all the dbs, by incrementing each id with 1 after being used.

```
4 references
private int GenerateId(string tableName)
{
    SqlConnection connection = new SqlConnection();
    SqlCommand cmd = new SqlCommand("SELECT TOP 1 * FROM " + tableName + " ORDER BY ID DESC ", connection);
    connection.ConnectionString = ConfigurationManager.ConnectionStrings["ConnectionWebService"].ToString();
    connection.Open();
    SqlDataReader dr = cmd.ExecuteReader();
    int idNr = 0;
    if (dr.HasRows)
    {
        while (dr.Read())
        {
            idNr = dr.GetInt32(0);
        }
    }
    return idNr + 1;
}
```

FUNCTIONALITATE- used to get a DateTable with all the books that have the title provided in the BookName or if any of the books titles have the string in the title.

```
[WebMethod]
0 references
public DataTable GetBooksByTitle(string bookName)
{
    DataTable books = GetAllBooks();
    var filteredRows = books.AsEnumerable()
        .Where(row => row.Field<string>("name").IndexOf(bookName, StringComparison.OrdinalIgnoreCase) >= 0)
        .ToList();

    DataTable dt = books.Clone();
    dt.Rows.Clear();
    foreach (DataRow d in filteredRows)
    {
        dt.Rows.Add(d.ItemArray);
    }
    dt.TableName = "Books";
    return dt;
}
```

FUNCTIONALITATE- used to get a DateTable with all the books that have the author provided in the authorName or if the author has the string in its name. Moreover, the GetUserId method is used in order to get the id of an author by its name.

```
[WebMethod]
0 references
public DataTable GetBooksByAuthor(string authorName)
{
    var authorId = GetUserId(authorName);
    SqlConnection connection = new SqlConnection();
    SqlCommand cmd = new SqlCommand("SELECT * FROM Books WHERE idAuthor = '" + authorId + "'", connection);
    connection.ConnectionString = ConfigurationManager.ConnectionStrings["ConnectionWebService"].ToString();
    connection.Open();

    SqlDataReader dr = cmd.ExecuteReader();
    DataTable dt = new DataTable();
    dt.Load(dr);
    dt.TableName = "Books";
    return dt;
}
1 reference
public int GetUserId(string authorName)
{
    SqlConnection connection = new SqlConnection();
    SqlCommand cmd = new SqlCommand("SELECT id FROM Users WHERE username = '" + authorName + "'", connection);
    connection.ConnectionString = ConfigurationManager.ConnectionStrings["ConnectionWebService"].ToString();
    connection.Open();
    object result = cmd.ExecuteScalar();

    if (result != null)
    {
        return Convert.ToInt32(result);
    }
    else
    {
        return -1;
    }
}
```

Chapter 4. Application Testing

4.1. Introduction

Welcome to the documentation of the web application testing phase for our project. In this chapter, we will delve into the crucial process of application testing and its significance in ensuring the reliability, functionality, and overall quality of our web application. As the development cycle nears completion, thorough testing becomes paramount to identify and rectify potential issues, enhance user experience, and deliver a robust and polished product.

4.2. Application Testing Methods

In the realm of software development, application testing plays a vital role in ensuring the quality, functionality, and reliability of web applications. Testing methods serve as a systematic approach to identify potential defects, assess the application's performance, and validate its compliance with specified requirements. In this section, we will explore some of the key testing methods employed in our project, providing you with a comprehensive understanding of their purpose and application.
There is: Functional Testing, Performance Testing, Security Testing, Usability Testing, Compatibility Testing, Regression Testing.
We mainly focused on functional testing.
Functional testing focuses on verifying that the application's functionalities perform as intended. It involves testing individual features, user interactions, and system behavior to ensure they align with the defined requirements. This method encompasses various

techniques such as unit testing, integration testing, and system testing. By systematically validating each functional aspect of the application, we can identify and rectify any discrepancies, ensuring a seamless user experience.

4.3. Testing Results

1.

Login 1:

Test Case Description: Login – Negative Test Case

Test Scenario ID	Login-1	**Test Case ID**
Test Case Description	Login – Negative test case	**Test Priority**

S.No	**Action**	**Inputs**	**Expected Output**	**Actual Output**	**Data Base**	**Test Result**
1	Launch application					Pass
2	Enter invalid username & any Password and hit login button	Email id : invalid@xyz.com Password: ******	Message: "NOT OK"	Message: "NOT OK"		Pass

2. Login 2:

Test Case Description: Login – Positive Test Case

Test Priority: High

Test Scenario ID	Login-1			**Test Case ID**		
Test Case Description	Login – Positive test case			**Test Priority**		
S.No	**Action**	**Inputs**	**Expected Output**	**Actual Output**	**Data Base**	**Test Result**
1	Launch application					Pass
2	Enter correct username & correct Password and hit login button	Username: dariald psd: ********bra	Message: "OK"	Message: "OK"		Pass

3. Register

Test Case Description: Login – Positive Test Case

Test Priority: High

Test Case Description	Register – Positive test case		**Test Priority**			
S.No	**Action**	**Inputs**	**Expected Output**	**Actual Output**	**DataBase**	**Test Result**
1	Launch application					Pass
2	Enter username, email, password, Password again, label	mariaD, maria@yahoo.com, parola3, parola3, 4	Message: "user added"	Message: "user added"	User added in place in DataBase.	Pass
3	Enter username, email, password, password again, label	ionP, ion@yahoo.com, parola4, parola 4, 5	Message: "user added"	Message: "user added"	User added in place in DataBase	Pass

Register

Test Case Description: Login – Negative Test Case

Test Scenario ID		**Test Case ID**	
Test Case Description	Register – Negative test case	**Test Priority**	

S.No	**Action**	**Inputs**	**Expected Output**	**Actual Output**	**DataBase**	**Test Result**
1	Launch application					Pass
2	Enter username, email, password, Password again, label	zuzuD, zuzu@yahoo.com, parola3, parola4279374293, 4	Message: **"passwords do not match!"**	Message: **"passwords do not match!"**	Nothing Added	Pass
3	Enter username, email, (password, password again), with the info *already existing in table*, label	ionP, ion@yahoo.com, parola4, 5	Message: "Username already exits"	Message: "Username already exits"	Nothing Added	Pass
4	Register with two different passwords	"Abracadabra", "abracadabra"	Message: "Passwords do not match"	Message: "Passwords do not match"	Nothing Added	Pass

Main app form

5. Home page "button" 1 – Positive test case

Test Case Description		Home page – Positive test case		Test Priority high		
Pre-Requisite				Post-Requisite		
S.No	Action	Inputs	Expected Output	Actual Output	DataBase	Test Result
1.	Search a book by "author". The book will be searched for in the database after "idAuthor" and display the book.	Author: "Tomoko Fuse"	The Book searched for:. "homedecorating" – Home Decorating with Origami"	The Book searched for: "homedecorating" – Home Decorating with Origami"	Searches in database after idAuthor,	passed
2.	Search a book by "title". The book will be searched for in the database after "pdfFile" and display the book.	Title: "analogcircuitdesign" – Analog Circuit Design	The Book searched for: "analogcircuitdesign" – Analog Circuit Design	The Book searched for: "analogcircuitdesign" – Analog Circuit Design	Searches in database after pdfFile.	passed
3.	Search a book by "none". The book will be searched for in the database and display the book.	None:	The Book searched for:	The Book searched for:	Searches in database after similarities in all id, pdfFile, idAuthor,	passed

Home page "button" 2 – negative test case

Test Case Description		Home page – Negative test case			
S.No	Action	Inputs	Expected Output	Actual Output	Result
1.	Search a book by "author" but the book is not in the DataBase.	Author: "xyzAbc"	The message "No books to be shown."	The message "No books to be shown."	Passed
2.	Search a book by "title" but the book is not in the DataBase.	Title: "abcXyz"	The message "No books to be shown."	The message "No books to be shown."	Passed
3.	Search a book by "none" but the book is not in the DataBase.	None: "aaaaa"	The message "No books to be shown."	The message "No books to be shown.":	Passed

4. Settings "button" 1 – Positive test case

Test Case Description		Settings – Positive test case		Test Priority high		
Pre-Requisite				Post-Requisite		
S.No	Action	Inputs	Expected Output	Actual Output	DataBase	Test Result
1.	Press on the "Change Password button."		The page with options to change your password.	The page with options to change your password.		passed
2.	Write the correct old password and correct new password two times.	Correct passwords: Old: a*** New:b***	The password changes successfully. The message: "Your password has been changed successfully."	The password changes successfully. The message: "Your password has been changed successfully."	Only the password field in the database changes, if it has the correct rights.	passed
3.	Press on the "Change email "button.		The page with options to change your email.	The page with options to change your email.		passed
4.	Write the correct old email and correct new email.	Correct email: Old: daria.2001@yahoo.com New:2001.daria@yahoo.com	The email changes successfully. The message: "Your email has been changed successfully."	The email changes successfully. The message: "Your password has been changed email."	Only the email field in the database changes, if it has the correct rights.	passed
5.	Press on the "Change Icon" button.		The page with options to change the icon.	The page with options to change the icon.		passed
6.	Select the new icon.		The new icon is selected.	The new icon is selected.	"idIcon" field gets modified in database.	passed
7.	Press on the "Delete Account" button.		The message "Are you sure you want to delete your account?" with options "Yes" and "No" appears.	The message "Are you sure you want to delete your account?" with options "Yes" and "No" appears.		passed

Settings "button" 2 – Negative test case:

Test Case Description		Settings – Negative test case		Test Priority high		
Pre-Requisite				Post-Requisite		
S.No	Action	Inputs	Expected Output	Actual Output	DataBase	Test Result
1.	Write the incorrect old password and correct new password two times.	Password: a***	The message: "Your old password is incorrect!".	The message: "Your old password is incorrect!".		passed
2.	Write the correct old password and incorrect new password one out of two times.	New password: "abcd123" Confirm new password: "abcd1234"	The message: "Your new passwords do not match!"	The message: "Your new passwords do not match!"		passed

Settings "button" 2 – Negative test case:

Test Case Description	Settings – Negative test case		Test Priority high			
Pre-Requisite			Post-Requisite			
S.No	Action	Inputs	Expected Output	Actual Output	DataBase	Test Result
1.	Write the incorrect old password and correct new password two times.	Password: a***	The message: "Your old password is incorrect!".	The message: "Your old password is incorrect!".		passed
2.	Write the correct old password and incorrect new password one out of two times.	New password: "abcd123" Confirm new password: "abcd1234"	The message: "Your new passwords do not match!"	The message: "Your new passwords do not match!"		passed
3.	Write the correct old password and new password same as the old one.	Old password: "abcdef123" New password: "abcdef123" Confirm new password: "abcdef123"	The message: "Your new password matches your old password!"	The message: "Your new password matches your old password!"		passed
4.	Write the incorrect old email and correct new email.		The message: "Your old email is incorrect!".	The message: "Your old password is incorrect!".		passed
5.	Write the correct old email and new email same as the old one.	Old: daria.2001@yahoo.com New: daria.2001@yahoo.com	The message: "Your new email matches	The message: "Your new email matches		passed

	the old one.		your old one!"	your old one!"		

5. **Add a book "button"**

Test Case Description	Add a book – positive test case		Test Priority			
S.No	Action	Inputs	Expected Output	Actual Output	Data Base	Test Result
1.	Press on the "Add a book" button.		The page where you can add a book opens.	The page where you can add a book opens.		passed
2.	Press on the "Browse " button.		List of existing books form database.	List of existing books form database.		passed
3.	Write a title for the new book.		A title will be given to the book. The title is put in the database in the name field from books.	A title will be given to the book. The title is put in the database in the name field from books.	New name: "Aventuri prin munti".	Passed
4.	Press on one of the images for the book cover.		The selected image is on the chosen book cover. Image attributed to one book in database.	The selected image is on the chosen book cover. Image attributed to one book in database.	The "idIconBook" field gets modified.	Passed.

6. **Logout**

Test Case Description	Logout – test case				
S.No	Action	Inputs	Expected Output	Actual Output	Test Result
1.	Press the "logout" button.		Getting logged out of the account and of the web page.	Getting logged out of the account and of the web page.	passed

7. **Your books, Your downloads.**

Will show information if they contain information.

Chapter 5. Conclusions

By skillfully integrating and harnessing the full range of functionalities we have previously presented, carefully considering their intricacy, meticulously refining their designs, and effectively managing the vast amount of data they manipulate, we have unequivocally accomplished our intended goal for this application.

Moreover, this project not only meets our immediate objectives but also leaves a tantalizing open-endedness that invites further exploration and expansion, promising a myriad of potential applications in different realms. Its adaptability and relevance in any social environment ensure that it can genuinely deliver the profound impact that our dedicated team believed to be essential.