



A Project Report on

Edusphere

An Efficient Classroom Management System

Course name: Object Oriented Programming (CSE - 2112)

Submitted by :

- 01. Waki As Sami [Roll: 11]
- 02. Zisan Mahmud [Roll: 23]
- 03. Md. Sadman Sakib [Roll: 49]

Submitted to :

Dr. Muhammad Ibrahim
Md. Ashraful Islam

Department of Computer Science & Engineering
University of Dhaka

Table of Contents

1. Introduction

[1.1] The Idea

[1.2] Initiatives

[1.3] Features

[1.4] Tools & Technologies

[1.5] Workload breakdown

[1.5.1] Zisan Mahmud

[1.5.2] Md. Sadman Sakib

[1.5.3] Waki As Sami

2. Requirement Analysis and objectives

[2.1] Purpose

[2.2] Intended Users

[2.3] Intentions

3. Design & Implementation

[3.1] UX Flow

[3.2] UI Showcase

[2.3] System Design

4. Project Features

5. Code Features

6. Discussion

7. Code Repository

8. Conclusion

[8.1] Challenges & Solution

[8.2] To sum everything up

Introduction

[1.1] The Idea

The cornerstone of any healthy society, education is essential to determining the course of both the individual and the global future. Technology developments have significantly altered how education is provided in recent years. One such example of how the conventional methods of controlling classrooms have changed is the use of classroom management systems.

Eduphere is an excellent classroom management system created to improve the learning process by giving students a full range of tools to efficiently run their studying. By using a single platform, the technology enables students to track progress, streamline administrative processes..

An overview of the Eduphere system's features and advantages for students are given in this report. It also talks about the difficulties encountered and how they were resolved during the development period. The paper ends with suggestions for enhancing the system in the future.

This report's overall goal is to give readers a thorough knowledge of Eduphere and how it could completely change how classroom management is done in the future.

[1.2] Initiatives

We saw the necessity for a compact management system for a student to manage his or her courses and the information pertaining to them. Our Edusphere project offers a solution to many of the problems we frequently encountered as students. We were inspired to create this application by the problems we had with the system we currently employ. the key problems are:

- All of the courses are fixed but we don't manage them efficiently
- Reading materials are not properly arranged by us as a student
- We don't use proper time management tools
- We don't think of keeping track of our results and attendances
- We lack a single platform to track our daily activities

Now to solve these problems, various other third-party apps like online CGPA calculator or google drive, docs have come into help, however a dedicated system for the students is still lacking. So our application designed with a welcoming and modern interface can be extremely beneficial to students

[1.3] Features

1. Distinct accounts to each student.
2. Allows students to sign up or sign in with proper authentication of the information provided.
3. Provides a secured database containing all the personal information of password-protected student accounts.
4. Provides a flexible and user-friendly dashboard to access the features of the portal.
5. Allows students to track their attendance and track their time efficiency
6. Provides a more structured and organized view of course materials
7. Allows students to track their semester result.
8. Allows students to keep note of their future actions.

[1.4] Tools and Technologies

1. UI/UX design : Adobe Creative Suite
2. Code IDE : IntelliJ with SceneBuilder
3. Code Base : JAVA

4. Framework : JavaFX
5. Database : MySQL
6. Design Code Base : CSS
7. UX Code Base : FXML

[1.5] Workload Breakdown

[1.5.1] Zisan Mahmud

1. Dashboard, JavaFX FXML and interface design
2. Books Collector
3. Attendance management
4. Graph controller

[1.5.2] Md. Sadman Sakib

5. Function of time
6. Result management
7. Subject and subject controller
8. UI And UX design

[1.5.3] Waki As Sami

9. Sign up
10. Log in
11. Keep Notes

Requirement Analysis & Objectives

[2.1] Purpose:

The main purpose of this project is to enable students to manage their day to day study work and improve their ways of learning . At present there aren't many usable tools or structured software which the students can use for various types of self study and improvement purposes. For this reason we planned to create a tool and platform from which they can improve and bring out their very best.

[2.2] Intended Users:

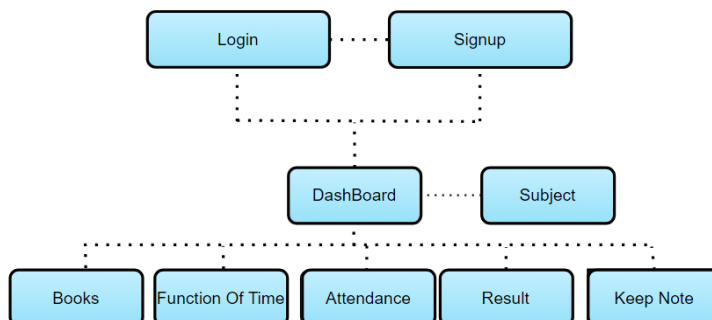
The project name suggests that it will be used by students or learners who are striving for competence in a subject or major.

[2.3] Intentions:

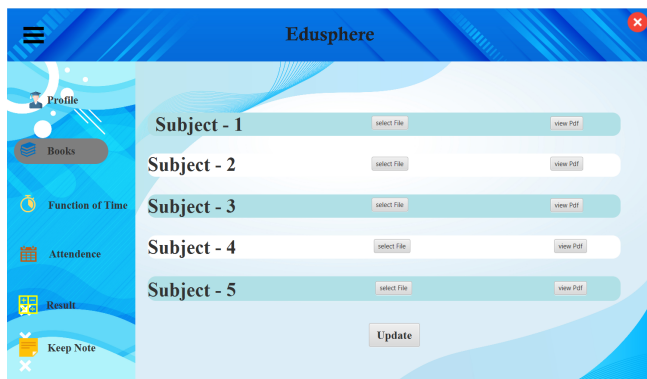
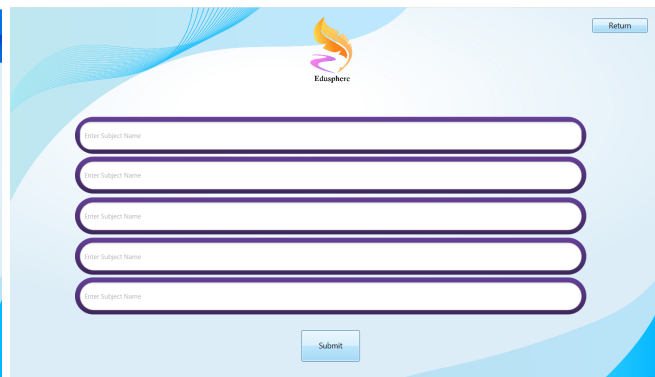
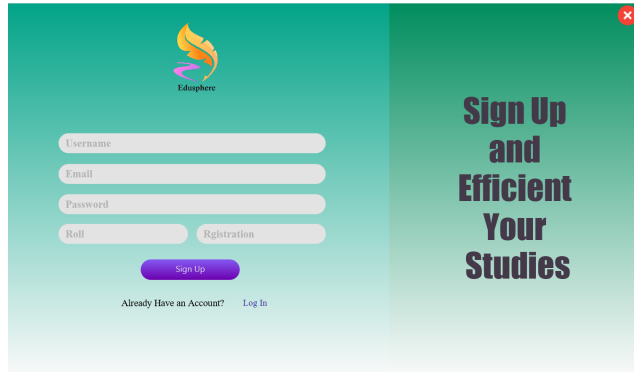
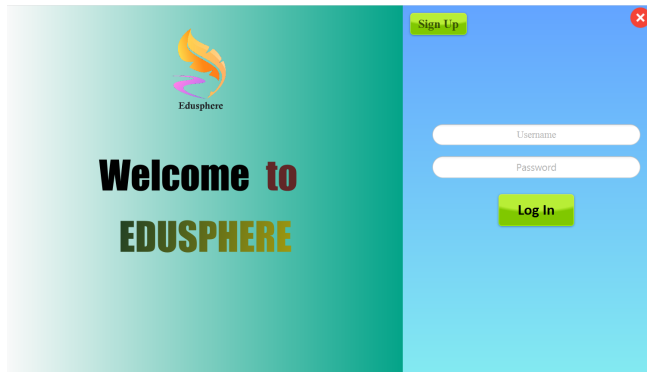
In this project, we want to demonstrate a complete "Study Management System". By using this project, a student can manage and manipulate his studies much more easily. For instance, managing track record data, notes, documents, getting routined notifications and other services, attendance tracking, study assistance system etc. will be handled in this project. In a nutshell, a complete management system will be found in this project by which sufferings will be lessened for both students as well as solo learners.

Design and implementation

[3.1] UX Flow



[3.2] UI Showcase



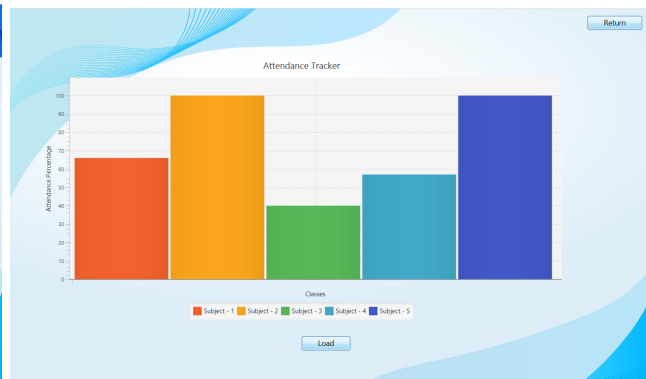
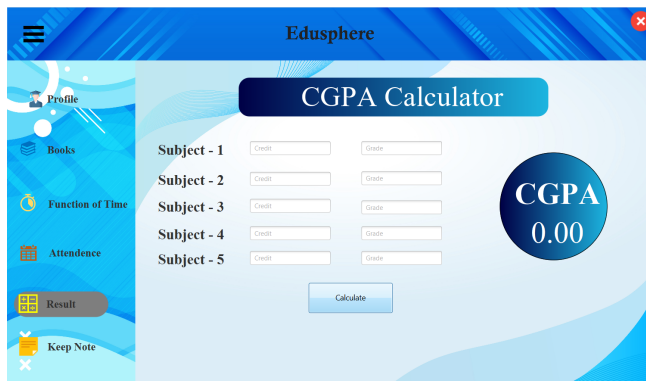


Edusphere

Profile
Books
Function of Time
Attendance
Result
Keep Note

Course Name	Class Taken	Class Attended	Percentage
Subject - 1	<input type="text"/> 0.0	<input type="text"/> 0	0.00%
Subject - 2	<input type="text"/> 0	<input type="text"/> 0	0.00%
Subject - 3	<input type="text"/> 0	<input type="text"/> 0	0.00%
Subject - 4	<input type="text"/> 0	<input type="text"/> 0	0.00%
Subject - 5	<input type="text"/> 0	<input type="text"/> 0	0.00%

Update Show More

Edusphere

Profile
Books
Function of Time
Attendance
Result
Keep Note

CGPA Calculator

Subject - 1 Credit: Grade:

Subject - 2 Credit: Grade:

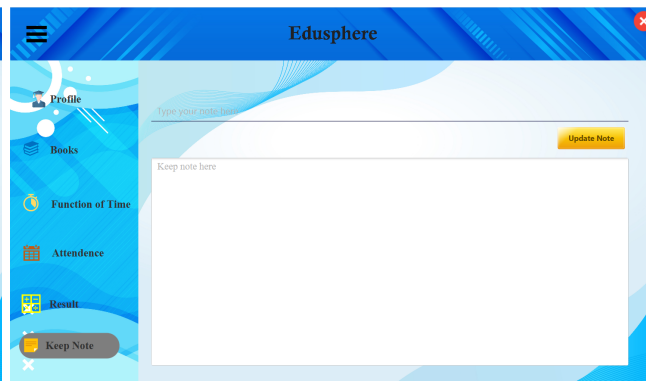
Subject - 3 Credit: Grade:

Subject - 4 Credit: Grade:

Subject - 5 Credit: Grade:

Calculate

CGPA 0.00



Edusphere

Profile
Books
Function of Time
Attendance
Result
Keep Note

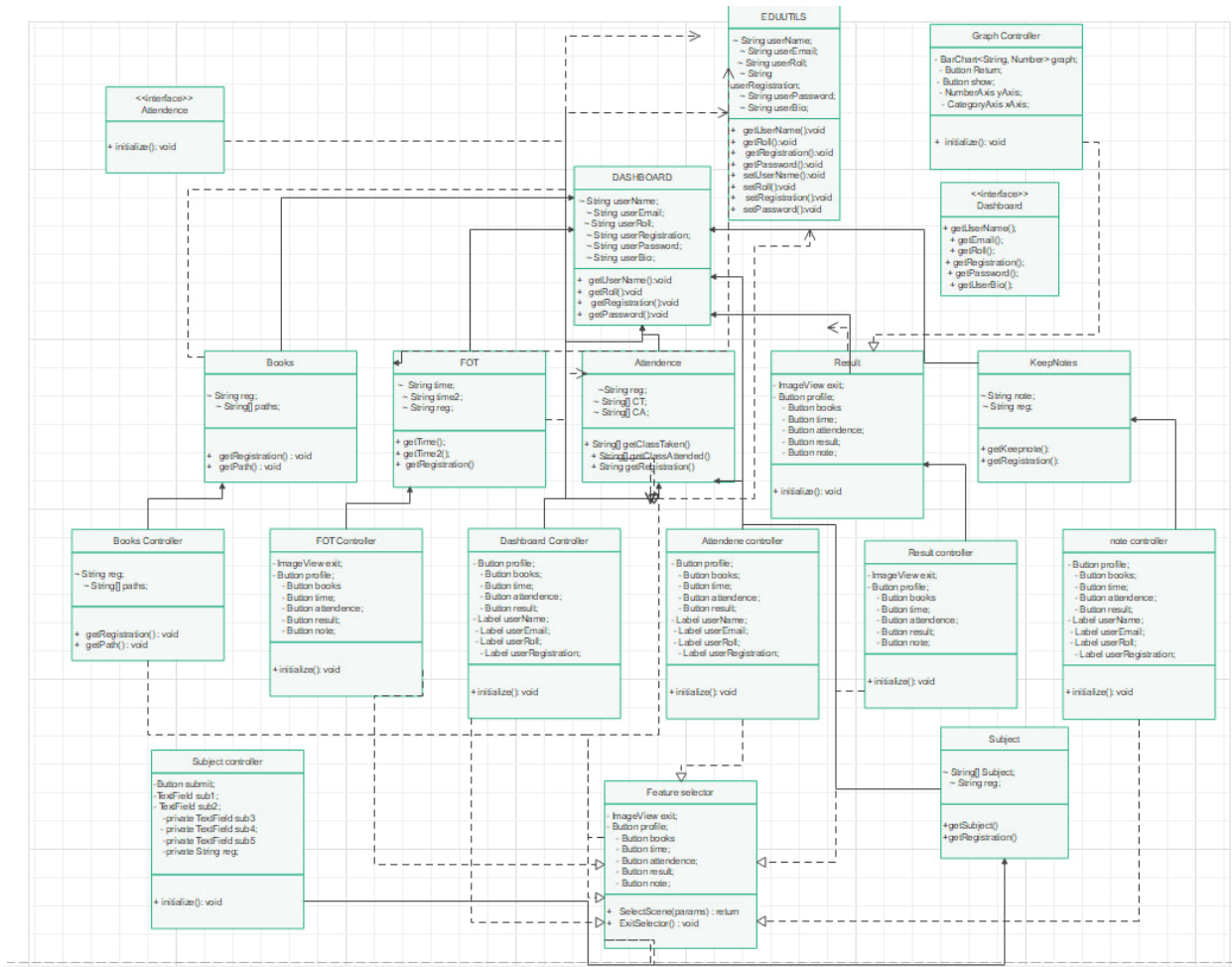
Type your note here

Update Note

Keep note here

[3.3] System Design

The entire codebase is separated into two main directories. The target folder for the FXML scenebuilder files and graphics, as well as the java folder, which contains all of the java files. The java folder is divided into two packages- one for classes and interfaces and the other for controllers . They are described below :



[3.3.1] Java Folder

- ProjectEdusphere

All the classes and interfaces are made in this package according to the UML class diagram. They are:

EDUSPHERE

This is the main driver class that will launch the program

EDUSPHEREUtils

This class is used to change scene and build connection with database

FeatureSelector

This class is used to navigate all the class

DashBoard, Books, Subject, FOT, Attendance, Result and Note

These classes are used to contain methods and variables for performing different operations.

- **Controllers**

This package contains the controller classes that are used to handle action events.

[3.3.2] Resources

- All the images, css files and FXML files are in this package.

[3.3.3] Implementation of Java Design Principles

1. DRY Principle

The DRY principle stands for the Don't Repeat Yourself principle. It is one of the common principles for all programming languages. The DRY principle says: "Within a system, each piece of logic should have a single unambiguous representation." During our project development, we tried to use Polymorphism and Inheritance in every possible case. This made our code easy to understand and simple for later development.

2. SRP Principle

SRP is another design principle that stands for the Single Responsibility Principle. The SRP principle says that in one class, there should never be two functionalities. In our project we use every single class for a unique responsibility and every class has a name similar to its functionalities.

3. LSP Principle

LSP is another design principle that stands for Liskov Substitution Principle. The LSP states that the derived class should be able to substitute the base classes without changing our code. In our code, the behavior of the superclass is

preserved in its subclasses, allowing for polymorphism and avoiding unexpected behavior.

4. DIP Principle

The Dependency Inversion Principle is maintained strictly in our codebase

5. KISS Principle

We followed the KISS principle(Keep it simple and stupid) to make our code easy to understand. This helps to find the bugs easily and easily to extend the project. We also followed suitable naming conventions.

Project Features

❖ Synchronized Multi-User Login System:

“Edusphere” offers a platform that integrates logins for several users onto one system.

❖ Secure Password:

As per our security policy passwords are stored in secured database which can be accessed through

❖ Storing data locally:

As per constraints networking attributes were not used and data was stored in local database

❖ User friendly interface:

Various instructions on usage helps user to communicate with the system efficiently

❖ GitHub WorkFlow:

Regularly updated and feature wise work done and project collaboration using github. Extensive commit and cautious merge across multiple platforms (Linux, Mac) made the project distinct.

Code Features

- Easy but compact Graphical User Interface (GUI).
- Simple and pretty easy to understand the code.
- Uses of Encapsulation, inheritance and polymorphism have been ensured which are the fundamental ideas of OOP language.
- By using encapsulation, we make the data or information much safer.
- By using Inheritance, we have made the best reuse of the code. As a result, we need not write any code more than once.
- Method overriding and method as well as constructor overloading are also used which are the basic concepts of polymorphism. In this case, we used the same name but with different prototype and functionality.
- Used setter and getter methods to set and get various types of data when needed.
- We used constructors to set the data.
- Well-documented code with Software engineering best practices implemented. Ideal for further development.
- Different controllers are implemented for different classes along with the FXML. The controller classes are used for loading the FXML and controlling the buttons and other intermediate functionalities.

Project Modules and discussion

All of Java's object-oriented features were utilized. The inheritance links are visible in our UML diagram. Here, the data is made available in a form that restricts it to a single class. Making classes like EdusphereUtils where all the required data will be stored allows for the most code reuse. Again, we avoided repeatedly querying the database by instead storing all the relevant information in a secure class and retrieving it using protected arrays. The codebase is organized into distinct modules where children classes can use super variables to transmit data to the parent classes.

Our main classes and interfaces are in the “src/main/java/com/example/projectedusphere”. Let's briefly discuss those classes and interfaces below:

- DsahboardInterface

This interface contains six abstract methods which can be inherited by subclasses. The DashboardInterface interface declares the following methods:

`getUserName()`: This method is expected to return a String representing the user's name.

`getEmail()`: This method is expected to return a String representing the user's email address.

`getRoll()`: This method is expected to return a String representing the user's roll (presumably referring to a specific role or position).

`getRegistration()`: This method is expected to return a String representing the user's registration information.

`getPassword()`: This method is expected to return a String representing the user's password.

`getUserBio()`: This method is expected to return a String representing a user's biography or profile information.

By providing these method signatures, the DashboardInterface defines a contract that classes implementing this interface must adhere to. Any class that implements the DashboardInterface interface will be required to provide concrete implementations of these methods, fulfilling the interface's contract.

The purpose of this interface is to define a common set of methods that can be used to access and retrieve various user-related information for a dashboard. By having this interface, it allows different classes to implement this interface and provide their own logic to retrieve and manage user data.

- Attendance, Boks, Dashboard, FOT, Notes, Result, SUBject

These classes are used to implement methods that are going to be used in the controller classes. The Dashboard class implements DashboardInterface. Other classes extend Dashboard class. The use of Dashboard class is to get the data from the database and hold the data in the variables declared in the class in run time. Other classes use the get method to get the data from the variables and use it.

- EDUSPHEREUtils:

This is a Java class named EDUSPHEREUtils with several static variables and getter/setter methods defined.

The static variables include:

- RegistrationNumber: a String to store the registration number of a user
- Email: a String to store the email address of a user
- Roll: a String to store the roll number of a user
- Name: a String to store the name of a user
- Password: a String to store the password of a user
- Bio: a String to store text for bio information of a user
- Note: a String to store text for notes taken by a user
- time: a String to store time information for a user
- time2: a String to store time information for a user
- Subject: a String array to store subject names for a user
- classtaken: a String array to store class names taken by a user
- classattended: a String array to store class names attended by a user
- Paths: a String array to store paths for files uploaded by a user

The getter/setter methods are defined for each static variable to allow other classes to access or modify their values.

Additionally, there is a static method named `getPaths()` and `setPaths(String[] Paths)` to get or set the `Paths` variable. Similarly, there are also other sets of getter/setter methods defined for the other static variables.

This class is used as a utility class to store and retrieve user information or data that needs to be accessed or modified across different parts of an application.

- **FeatureSelector**

This is a Java class for an event handler that is used to switch between different scenes in a JavaFx GUI application when different buttons or images are clicked.

- The `SelectScene()` method takes various buttons, an image view, and an optional `Object` parameter for each stage of the scene, as input.
- Each button is associated with an event handler that is invoked when it is clicked by the user.
- When each button is clicked, `EDUSPHEREUtils.changeScene()` method is called which takes several arguments, including the current event, the name of the next scene to switch to, the title of the new scene, the width and height of the new scene.
- The `EDUSPHEREUtils` class is another utility class that provides various methods including this `changeScene()` method.
- The `ExitSelector()` method is a standalone method that is used to add an event to the image view exit. When this image view is clicked, it will exit from the application using `Platform.exit()` method.

Our controller classes are in the “`src/main/java/Controllers`”. Let's briefly discuss those classes and interfaces below:

- **AttendanceController**

This class is used to handle attendance-related activities in a JavaFX GUI application. The `AttendanceController` class extends the `Attendance` class and implements the `Initializable` interface.

The FXML annotation is used to inject FXML elements such as Button, Label, and ImageView into the code. There are also a few static arrays and instance variables that represent attendance data for each subject.

In the initialize method, the `SelectScene()` method is used to handle various button clicks, including the exit image view. Additionally, the attendance data is loaded and displayed for each subject. The update button is used to calculate the attendance percentage for each subject, and the graph button is used to display an attendance graph.

Overall, this code provides a user interface for recording and displaying attendance data for each subject.

- BooksController

This is the controller class for a JavaFX GUI application that manages books functionality. The application allows the user to select and view PDF files for different subjects using file chooser dialogs.

The code sets various GUI components like buttons and labels using the `@FXML` annotation, which sets them as accessible to the controller class. It also initializes variables like the paths to different PDF files and the user's registration number.

The `initialize()` method is called when the controller is first loaded, and it sets up the GUI components based on the user's saved information, as retrieved through the `EDUSPHEREUtils` class. Specifically, it sets the names of the subjects and paths to the PDFs using various setter methods.

The `setPaths()` and `setreg()` methods set the paths and registration number, respectively, and the `getPaths()` and `getRegistration()` methods retrieve them.

The `selectFile()` and `viewPdf()` methods are event handlers for when the user selects and views a PDF file, respectively. When the user clicks the `selectFileButton` or any of its variants, a file chooser dialog is shown allowing them to select a PDF file from their device. The selected file's path is then saved to a string. If the `viewPdfButton` or one of its variants is clicked and a path to a PDF has been previously selected, the application attempts to open the PDF with the default PDF viewer on the user's computer.

Finally, the `update()` method also sets an event handler for when the user clicks the update button, in which case `EDUSPHEREUtils`' `'BOOK'` method is called with the event, the paths to the selected PDF files, and the user's registration number as arguments.

- **DashboardController:**

This code is for a controller class for a JavaFX GUI application that manages the dashboard functionality. The dashboard displays information about the user and allows them to edit their account settings, specifically their bio.

The `@FXML` annotation is used to set various GUI components like buttons, labels, and text fields like `profileBioinput`.

The controller extends the `DashBoard` class, which contains a constructor and various getter and setter methods. The constructor uses the `super` keyword to call the parent class's constructor and sets instance variables based on parent class attributes like the user's name, email, roll, registration number, password, and bio. Getter and setter methods are defined for each instance variable.

The `initialize()` method is called when the controller is first loaded, and it sets up the GUI components based on the user's saved information, as retrieved through the `EDUSPHEREUtils` class. Specifically, it sets the names and values of various labels with the appropriate data through the getter methods for each attribute. It

also adds event handlers for buttons like saveButton, sub, and logout that call various methods of the EDUSPHEREUtils class to change scenes and update or retrieve user data.

The handle() method for saveButton gets the text from profileBioinput and saves it to the BIO instance variable. It passes this information along with the registration number to the EDUSPHEREUtils method bio() to update this information for the user.

Finally, the profileBio label is updated to display the user's new bio using the setText() method, and the sub and logout buttons are given event handlers to change scenes to related functionality.

- FOTController

This class defines a controller class for the FOT (Focus on Time) functionality of an educational platform. The FOT functionality allows users to track their study time and set goals for their study sessions.

The class extends the FOT class and implements the Initializable interface. It contains several @FXML-annotated fields that correspond to GUI components like buttons, labels, and date pickers. The initialize() method sets up event handlers for the savebt button and calls the SelectScene method of the FeatureSelector class to handle scene navigation.

The handle() method for the savebt button calculates the user's remaining study time based on the difference between the current time and the end date set by the user. It gets the user's study time for the current session from the studytime text field and calculates the remaining study time. It then calls the FOT() method of the EDUSPHEREUtils class to update the FOT data for the user.

The method then calculates the user's study progress as a percentage of their total FOT goal. If the user has exceeded their FOT goal, an Alert dialog is

displayed to inform them of this. Otherwise, the totalHours, remainingtime, and progress labels are updated to display the user's FOT data.

Overall, this class provides users with a simple way to track their study time and set goals for their study sessions within the educational platform.

- **GraphController**

This class defines a controller class for the attendance graph functionality of an educational platform. The Attendance graph functionality shows students their attendance over time as a bar chart.

The class implements the Initializable interface and contains several @FXML-annotated fields that correspond to GUI components like buttons and bar charts. The initialize() method sets up event handlers for the Return and show buttons and calls the changeScene() method of the EDUSPHEREUtils class to handle scene navigation.

The handle() method for the show button gets attendance data for the user using EDUSPHEREUtils methods like getSubject(), getClasstaken(), and getClassattended(). It then calculates the attendance percentage for each subject and stores them in an array temp4. It sets up the bar chart with the subject names on the x-axis and attendance percentages on the y-axis.

The method then creates XYChart.Series objects for each subject and adds XYChart.Data objects containing the attendance percentage for each subject to the respective XYChart.Series object. Then, these series objects are added to a ObservableList data object, which is then set as data for the graph.

Overall, this code provides users with a simple and intuitive way to view their attendance data over time as a graph within the educational platform.

- **LogInController**

This class defines the controller class for log in functionalities for our project. This class implements Initializable interface and contains several @FXML-annotated fields that correspond to GUI components. The two handle() method for two buttons controls the login and sign up feature.

- NoteController

This class defines the controller class for Keep Note functionalities. This class implements Initializable interface. The handle() method under updatebutton takes the text from text field, save it on database and shows the text on text area.

- ResultController

This class defines the controller class for CGPA calculator functionalities that implements Initializable. The handle() method for calculateButton takes input the grade and credit and performs calculation on the inputs and gives output the aggregate CGPA.

- SignUpController

This class defines the controller class for sign up functionalities. Here the handle() method under the login button changes the FXML file to Login and the handle() method under the signup button takes the data input and stores the data in the database.

- SubjectController

The SubjectController class handles events and manipulates data related to subjects and the registration. It interacts with the UI elements defined in the associated FXML file and uses utility methods from EDUSPHEREUtils to manage scene transitions.

The SubmitSubject() method retrieves the text entered in the TextField objects, populates the subject array, and calls EDUSPHEREUtils.Subjects() passing the event, subject array, and registration. It then calls changeScene() to switch to the "Dashboard.fxml" scene with the title "Profile".

Code Repository

Github Link: <https://github.com/zisan23/Edusphere>

Conclusion

[3.1] Challenges and solution

- We faced difficulties while trying to manage too many button event handling with various conditions set on them
- A few features like animation and live dashboard weren't implemented due to constraints
- Due to time constraints we weren't able to add more features like centralized database and student connect features

[3.2] To sum everything up

In future we want to add admin control and more security on data. We also would like to make more flexibility in subject choice, feature to add more content and notes, self study time schedule and exam management and also we have a plan to connect students by creating a community portal.