School Management System

1Dharma Thanishq Nimmala ([nimmala.d@northeastern.edu](mailto:nimmala.d@northeastern.edu)) , 2Srinivasa Pavan Ganesh Vasa ([vasa.sr@northeastern.edu](mailto:vasa.sr@northeastern.edu)) , 3Rushikesh Karwankar ([karwankar.r@northeastern.edu](mailto:karwankar.r@northeastern.edu))

1Information Systems, Northeastern University  
2Information Systems, Northeastern University  
3Information Systems, Northeastern University

***Abstract*—** **The Student Management System is a software application developed to simplify the management of student records in educational institutions. Its main objective is to provide a user-friendly interface for teachers and administrators to manage student records efficiently. The system consists of modules for student registration, course management, departments management, faculty management and overall statistics. The student registration module allows administrators to manage student records, including personal information, academic records, attendance records, and any other relevant data. The course management module enables teachers to manage their courses, including creating course schedules, assigning assignments, and monitoring students' progress. The statistics module provides an overview of the student capacity per department, student vs courses, and other relevant statistics.  
The System also consists of student module which provide students with information about registered courses, personal student details, also allows themselves to mark their own attendance for current working day and provides an academic calendar with important dates. This program implementation majorly focused on Class Definition, Inheritance and Polymorphism, Interfaces, Collections and Iterators, Lists, Sets. During the development process, the team followed the Agile methodology, which allowed for continuous feedback and iterative development. This approach ensured that the system met the requirements of stakeholders, and any issues or concerns were addressed promptly. The outcomes of the project include an efficient and user-friendly system for managing student records, which can be easily customized to meet the specific needs of different educational institutions. The system can help to streamline administrative tasks, reduce errors, and improve the accuracy of student records, making it a valuable tool for educators and administrators. Additionally, the project can serve as a basis for future research and development in the field of educational technology.**

***Keywords -*** (Dashboard ,OOPS concepts, Inheritance and Polymorphism, Interfaces, Collections and Iterators, Lists and Sets, FileI/O)

# **I. Problem Description**

The current manual process of managing student records in educational institutions is time-consuming and prone to errors. The need for an efficient and reliable system that can manage student records has become a necessity. The purpose of this project is to design and develop a Student Management System using JavaFX that can automate the process of managing student records in educational institutions. The system should provide an easy-to-use interface that allows administrators and teachers to manage student records efficiently. This system should be capable of storing and managing student records, including course management, and other relevant information. The system should also be able to generate reports that can be used for decision-making purposes. The successful implementation of this project will lead to an efficient and reliable system for managing student records in educational institutions.

# **II. Analysis (Related Work)**

Several studies have been conducted in the field of educational technology to address the challenges of managing student records in educational institutions. According to a study by Althobaiti. [1], traditional paper-based systems are inefficient, prone to errors, and time-consuming. Therefore, digital solutions, such as student management systems, have been developed to address these challenges.  
Many student management systems have been developed and implemented in various educational institutions. A study by Mohammed et al. [2] explored the effectiveness of a student management system in managing student records, and it was found to be effective in reducing the workload of teachers and improving the accuracy of records. Another study by Singh and Gupta [3] proposed a student management system that included modules for student registration, course management, and grading. The system was found to be effective in improving the efficiency of administrative tasks.  
However, some existing solutions have limitations. For example, a study by Kuppusamy et al. [4] found that some student management systems lack user-friendly interfaces, making them difficult to use for teachers and administrators. Additionally, some systems do not provide sufficient customization options, making it challenging to meet the specific needs of different educational institutions.  
To address these limitations, the Student Management System proposed in this project is designed to be user-friendly and customizable. It includes modules for student registration, course management, and course statistics, providing a comprehensive solution for managing student records. The system's design and development follow industry-standard software development methodologies and utilize modern technologies, making it efficient, reliable, and secure.  
In conclusion, the development of the Student Management System is a significant contribution to the field of educational technology. It addresses the challenges of managing student records in educational institutions and offers a user-friendly, efficient, and customizable solution.

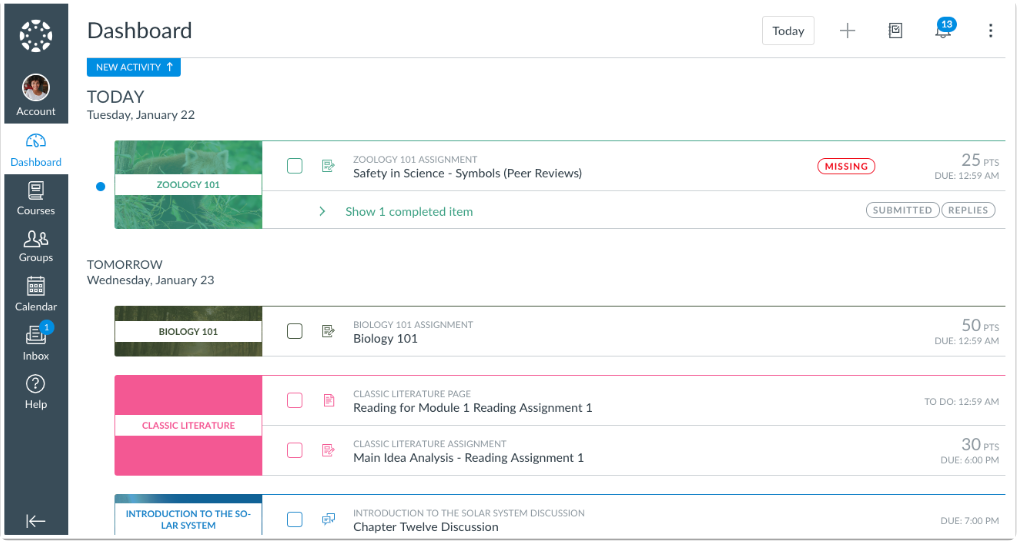


Figure 1. Screenshot of a tradition school management system (Canvas).

# **III. System Design**

User Interface: The user interface is designed using JavaFX. It includes a login screen for users to access the system and a dashboard where students' data can be displayed and managed.

Text file storage: The system stores data in text files on the local disk. These files can be easily read and written using Java's file handling APIs.

Data Model: A Java class will be created to represent a student. The class will have properties like name, ID, department, and other necessary information.

Data Entry: The system will allow users to add new students, and courses. When a user adds a new student or courses, the information will be written to a text file.

Data Display: The dashboard will display a list of all students, faculty, courses and departments in the system. When a user selects a student, their information will be read from the text file and displayed on the screen.

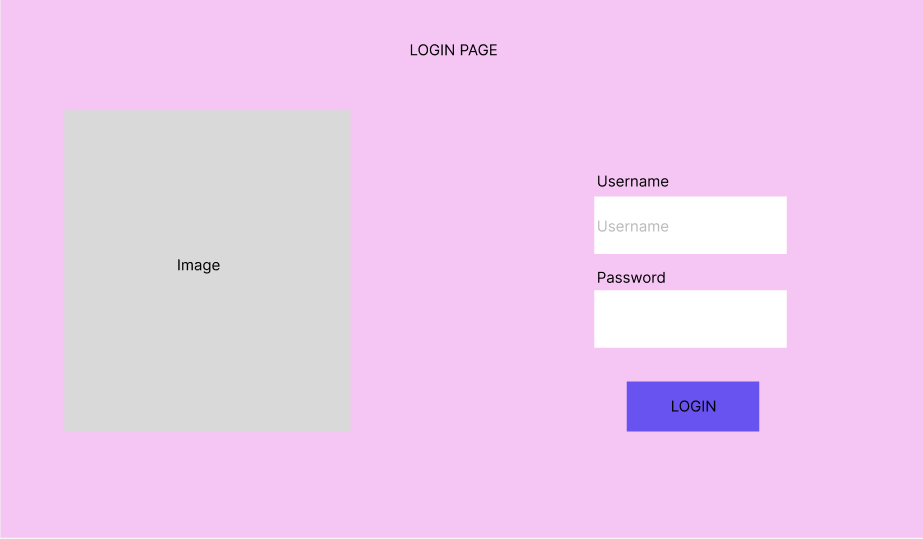
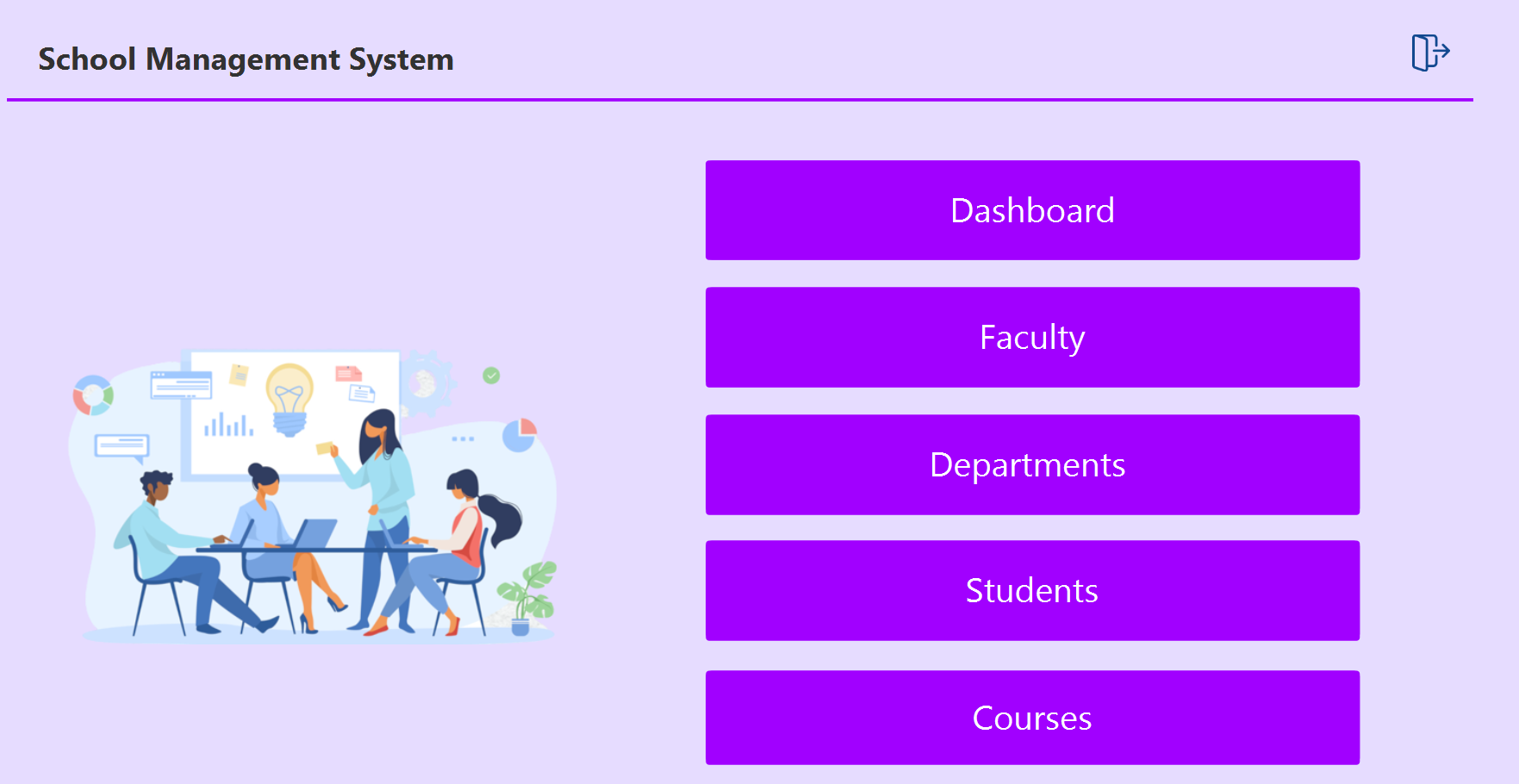
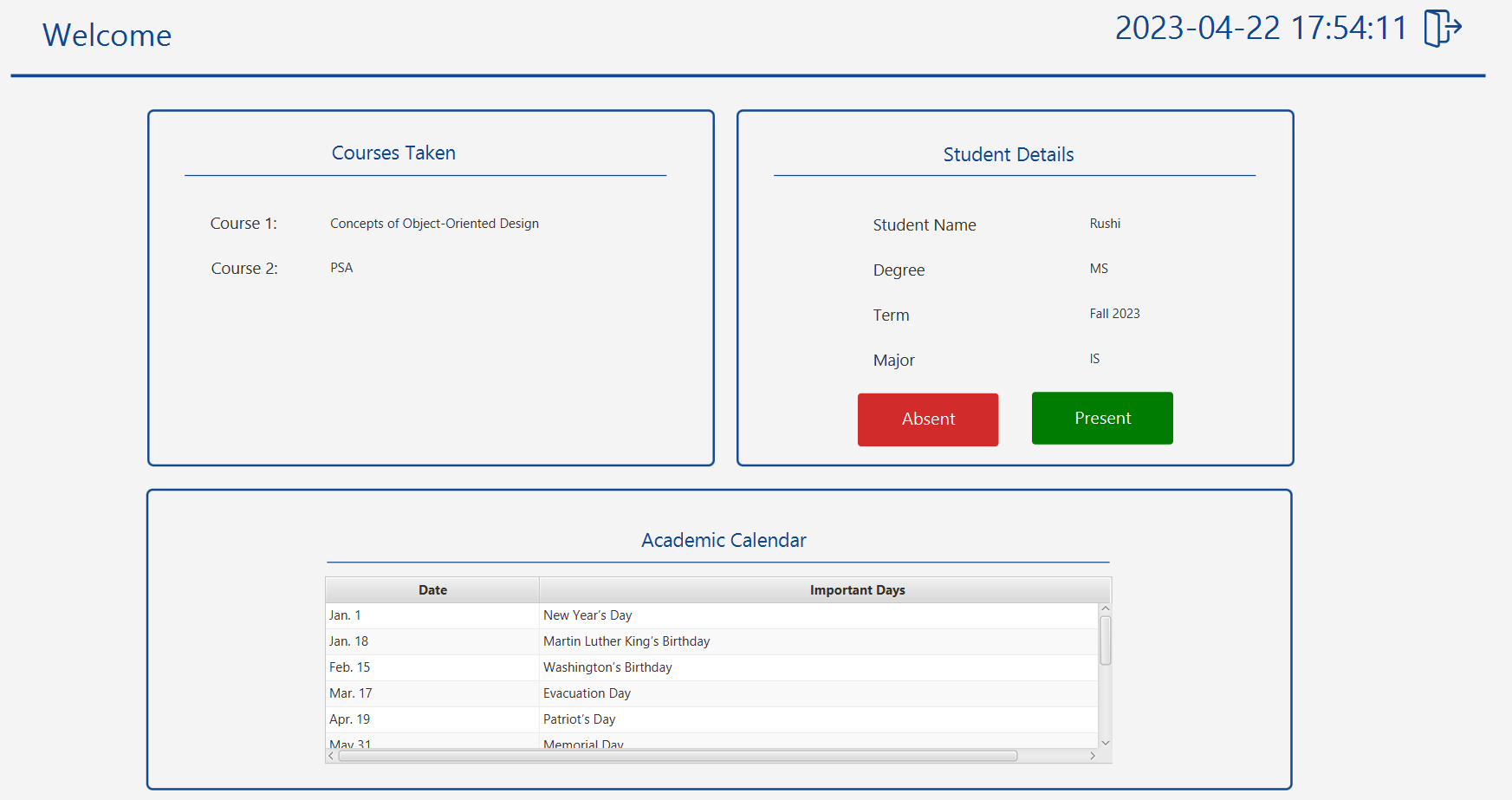
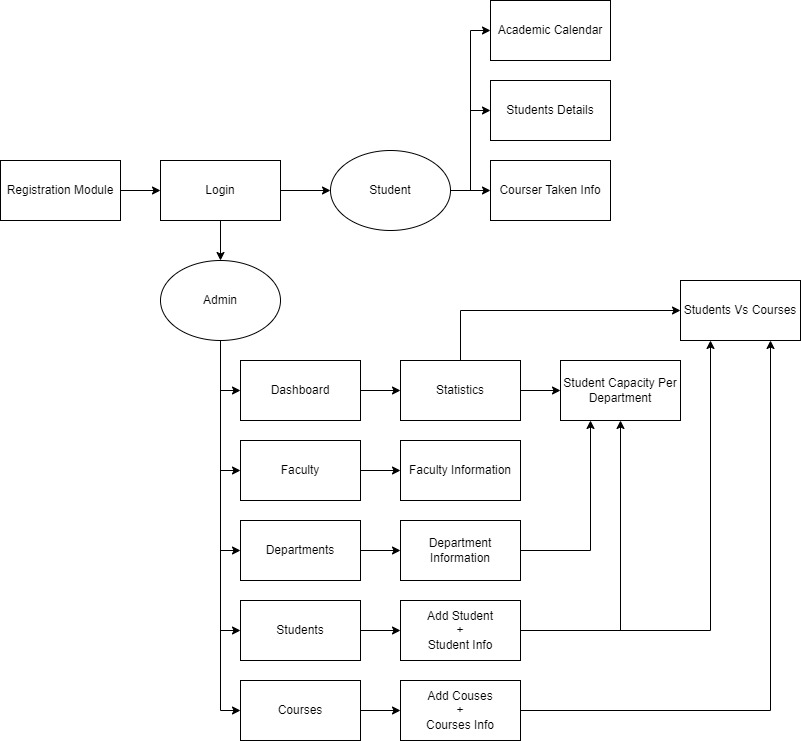
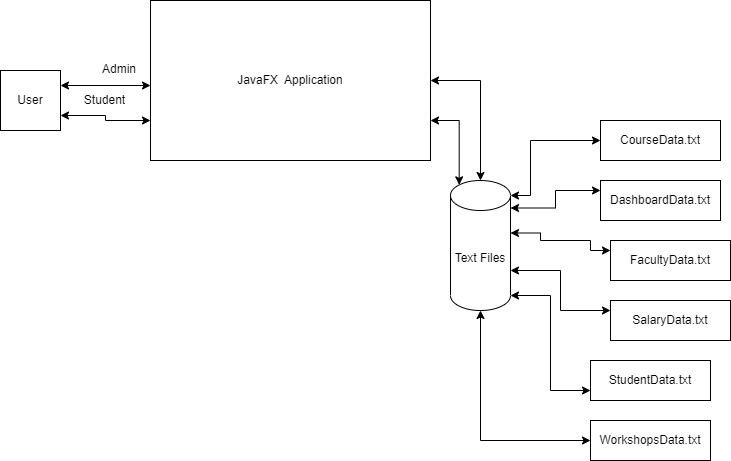
  
  
  
  


Figure 2. Draft UI designs(Figma Designs)

Figure 3. Work Flow Diagram   
  
  
Figure 4. System Architecture

# **IV. Implementation**

We have 6 sub-folders, namely controllers, data, fxml, model, home, images.  
**Controllers:** This section contains all the required controllers for the UI pages. Like whenever a button click is performed the action which is to be implemented in programmed in these controllers.  
Data: This section contains the logic to process all the data from our db i.e text files. For each entity like student, faculty etc. we have separate file which processes the reading the existing data from txt file and writing any newly created model into that file.  
Fxml: This section contains all the fxml files generated from the scenebuilder.  
Model: This folder contains all the entity objects that we use throughout our project like the student.  
Home: This sections contains the main.java file which acts as a entry point to our program.  
Images: All the images used in the UI are stored in this folder.

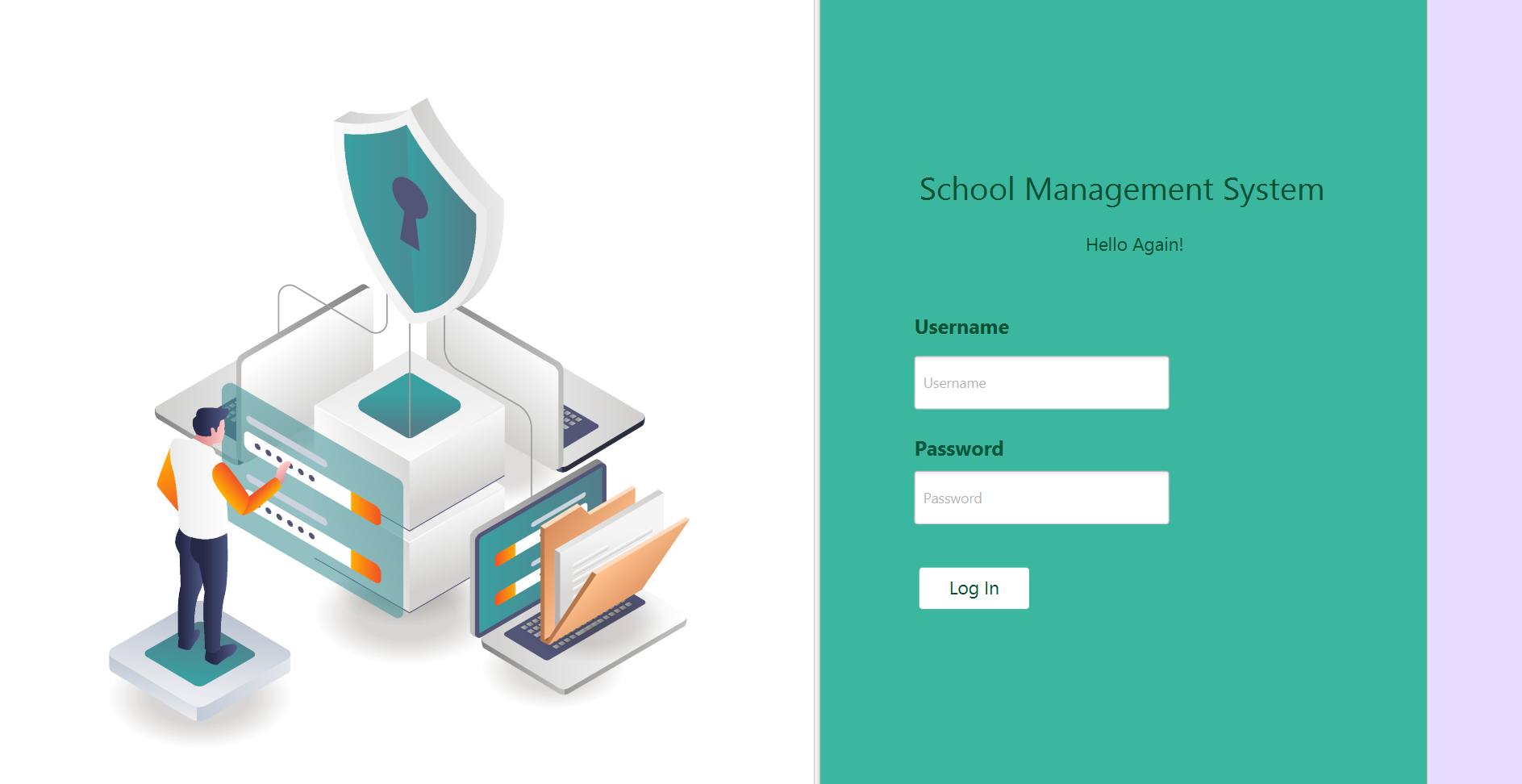
**A. C*ontrollers***

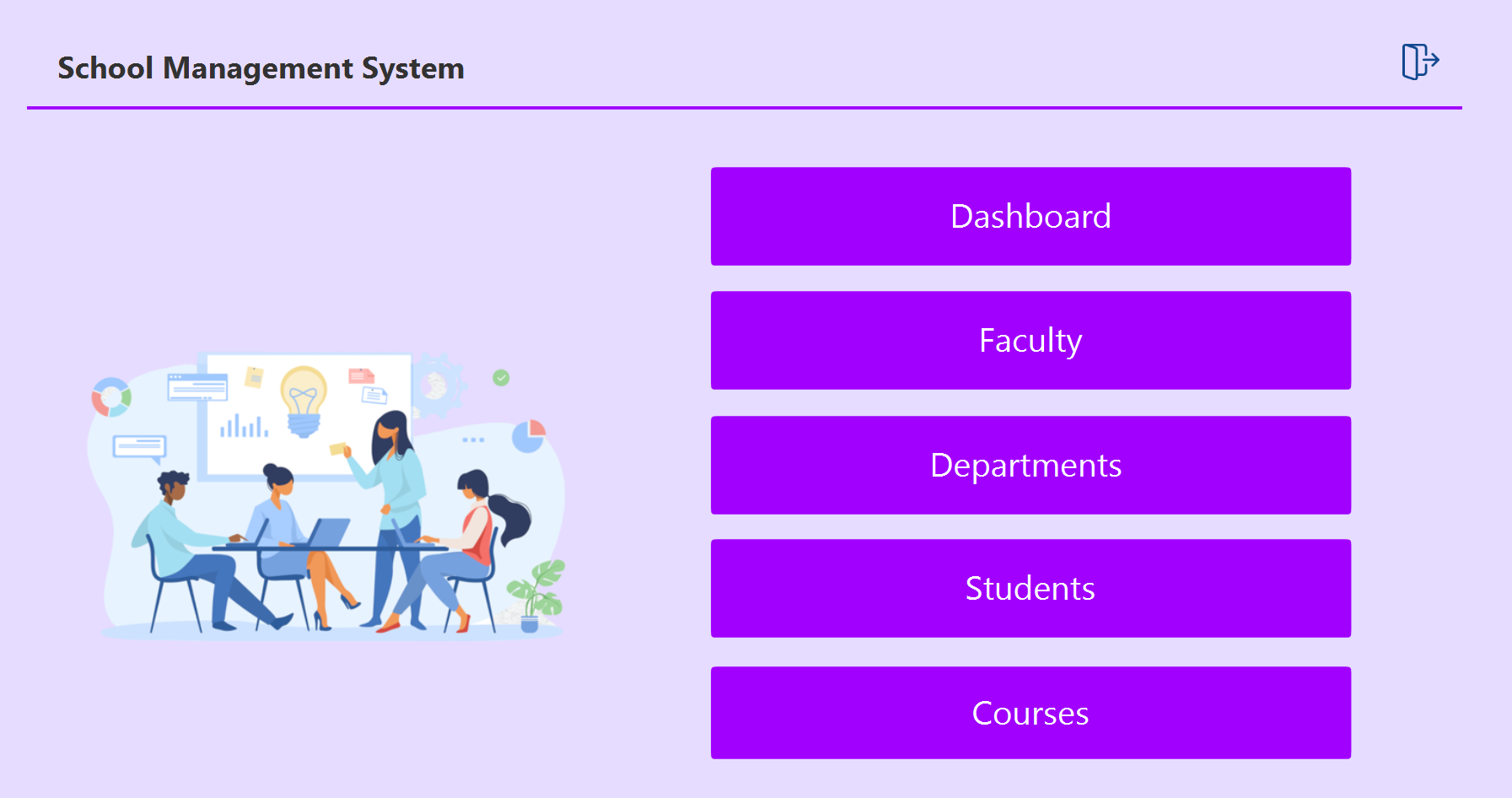
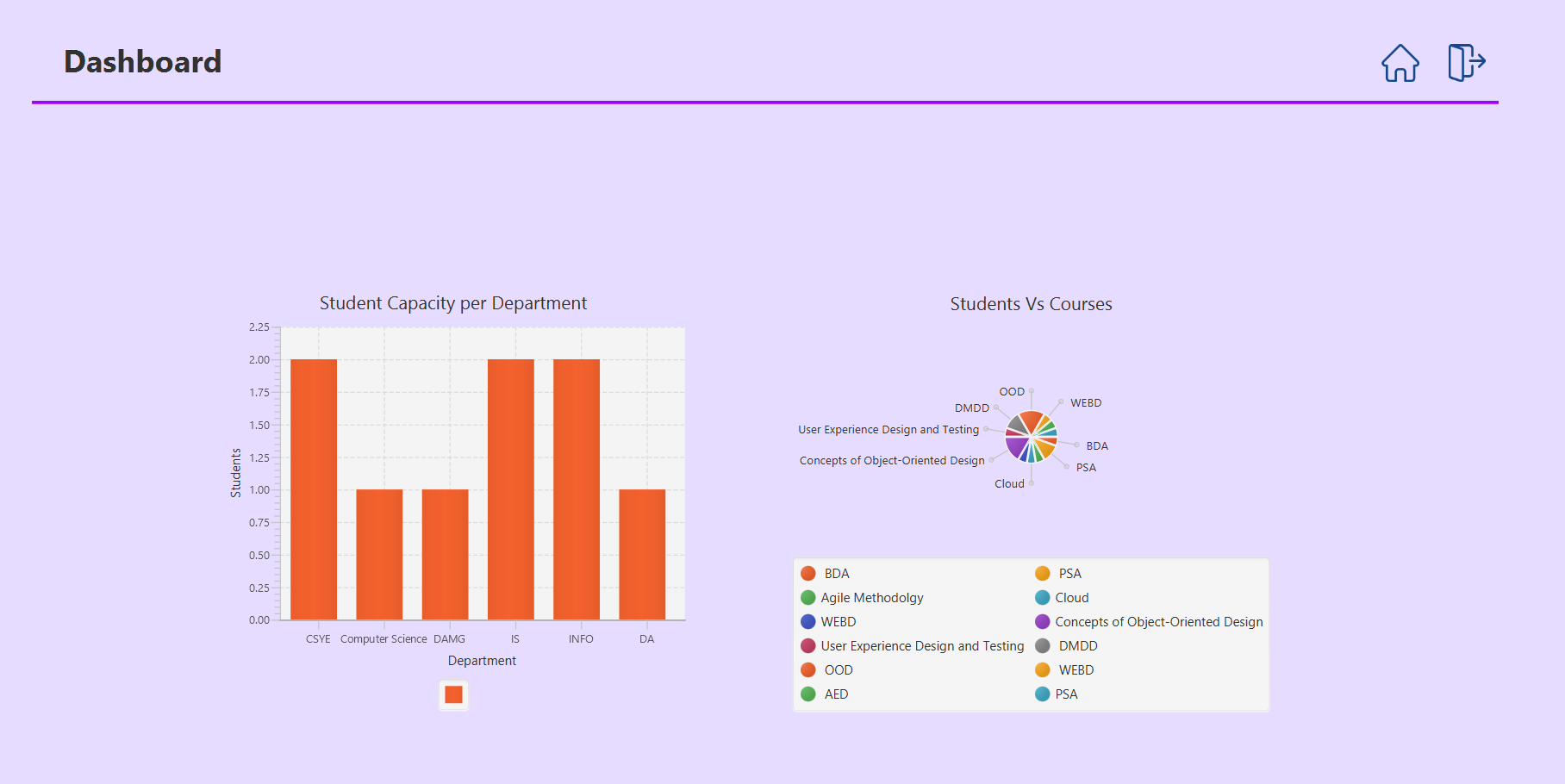
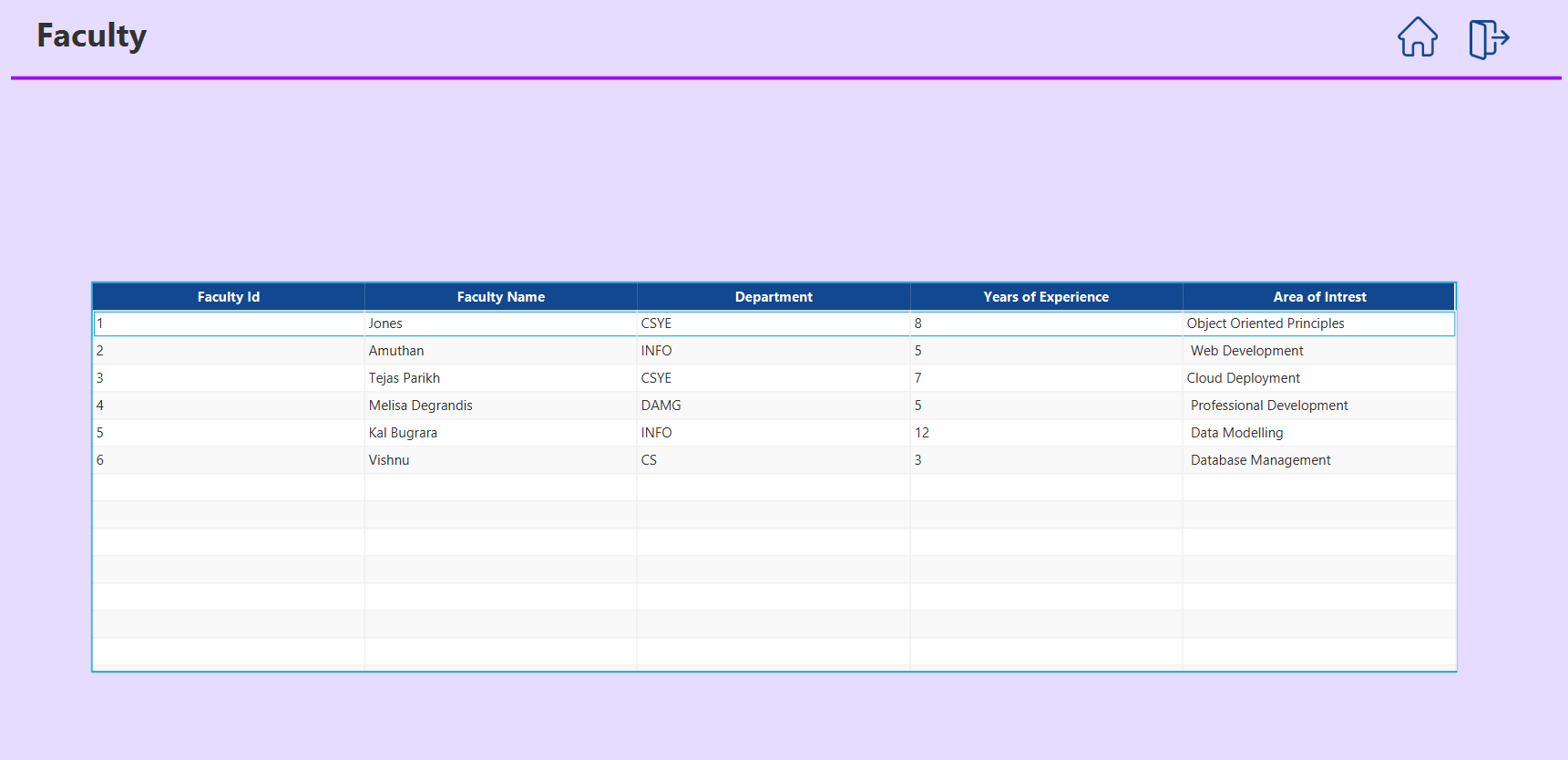
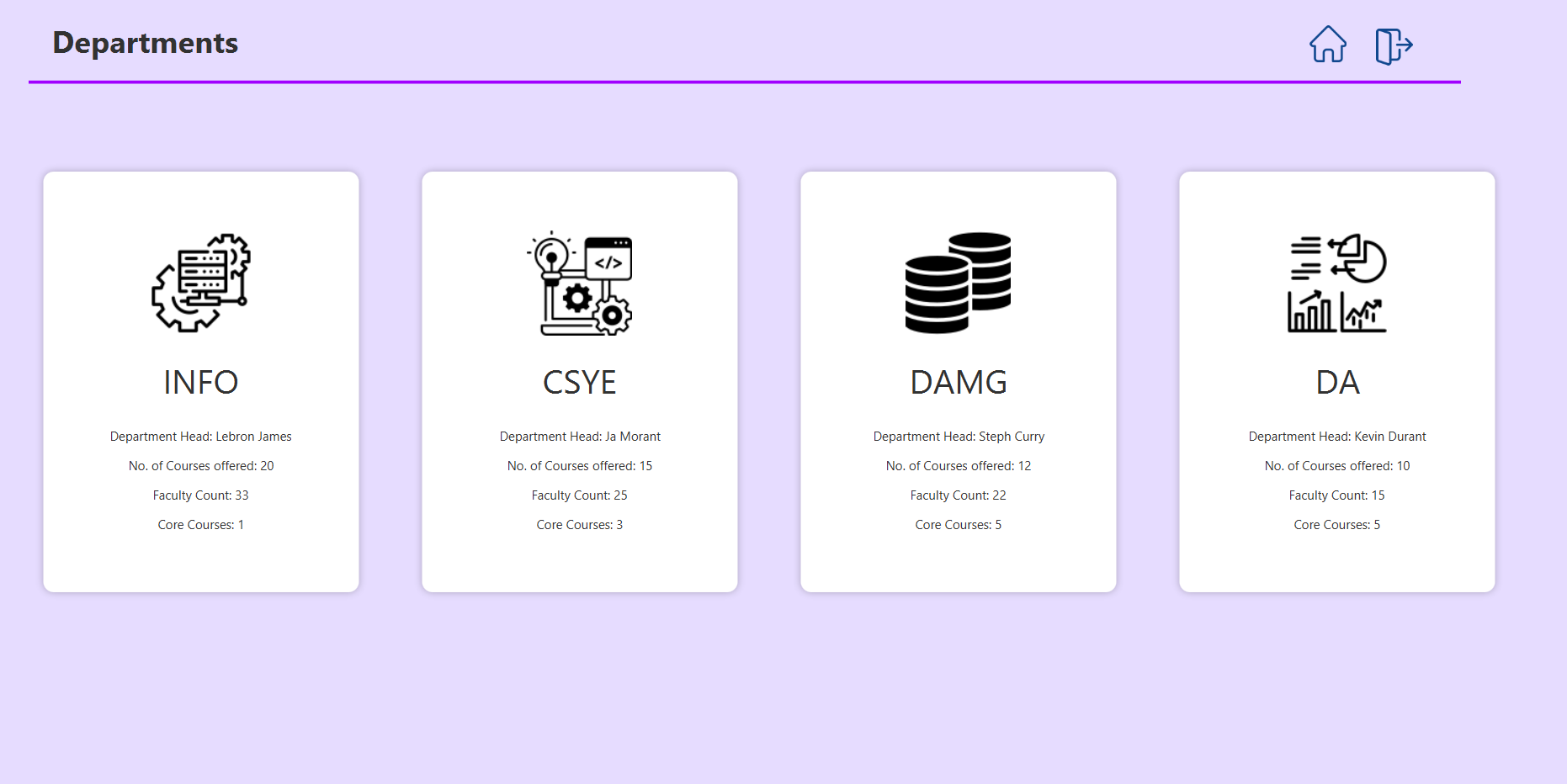
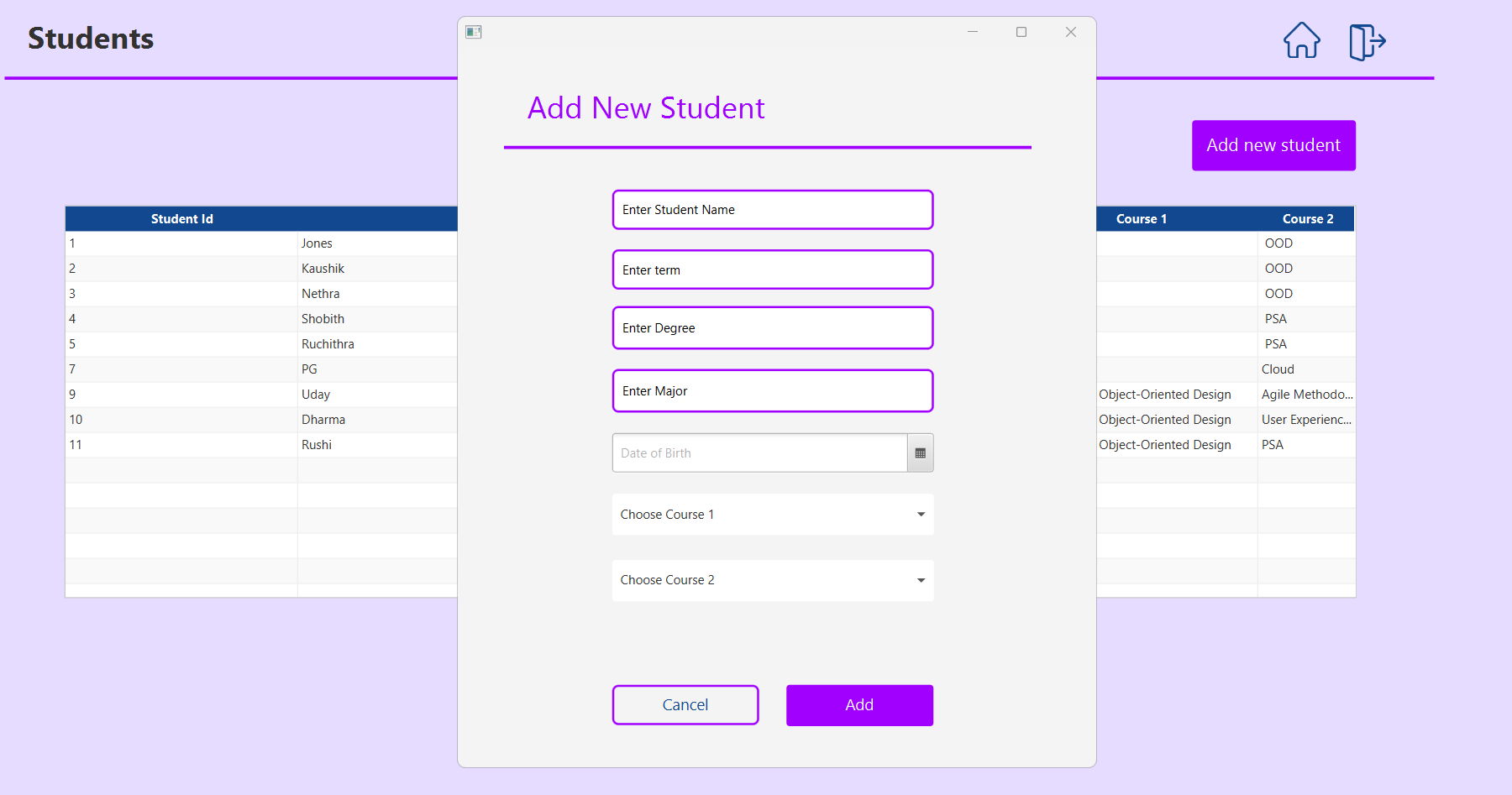
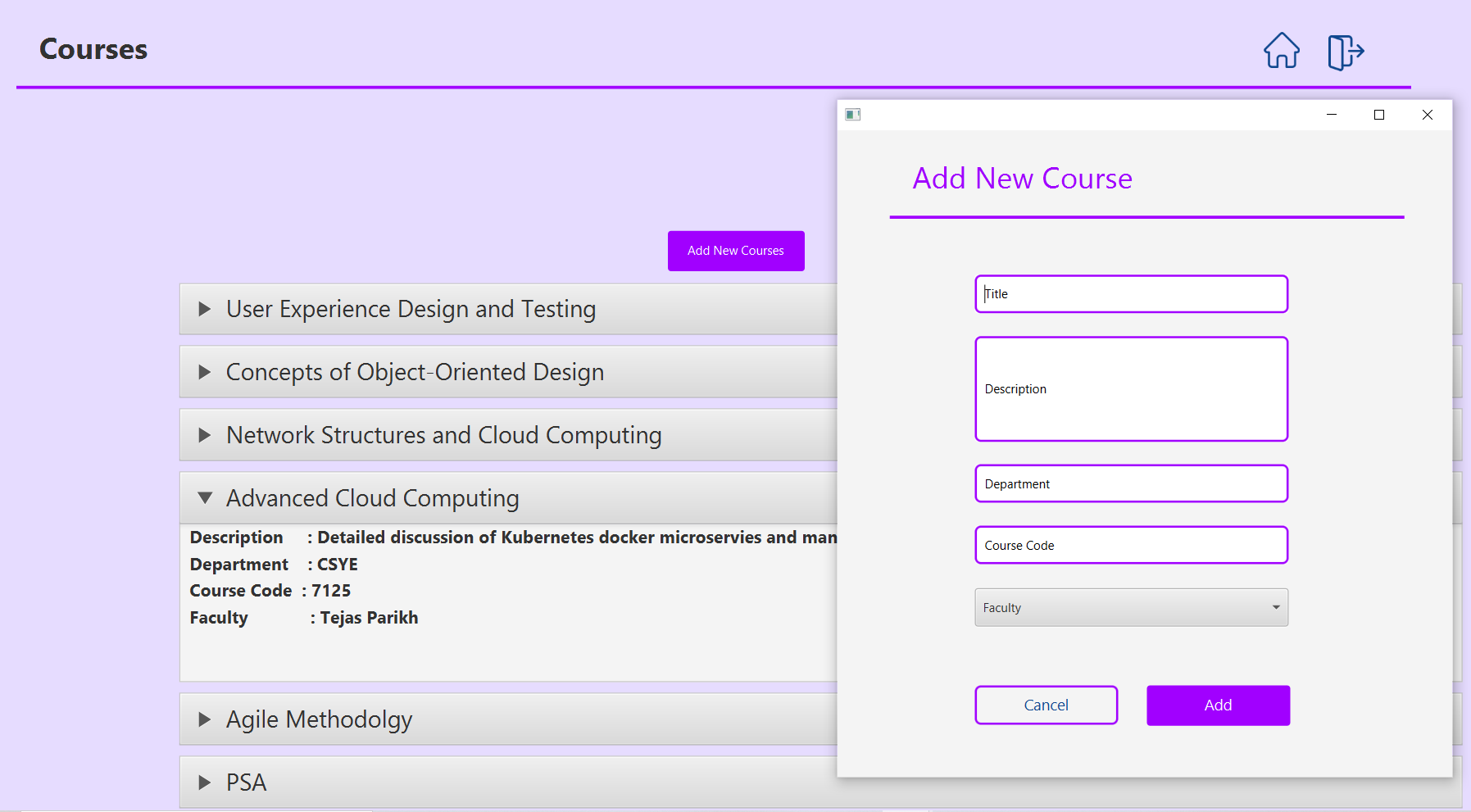
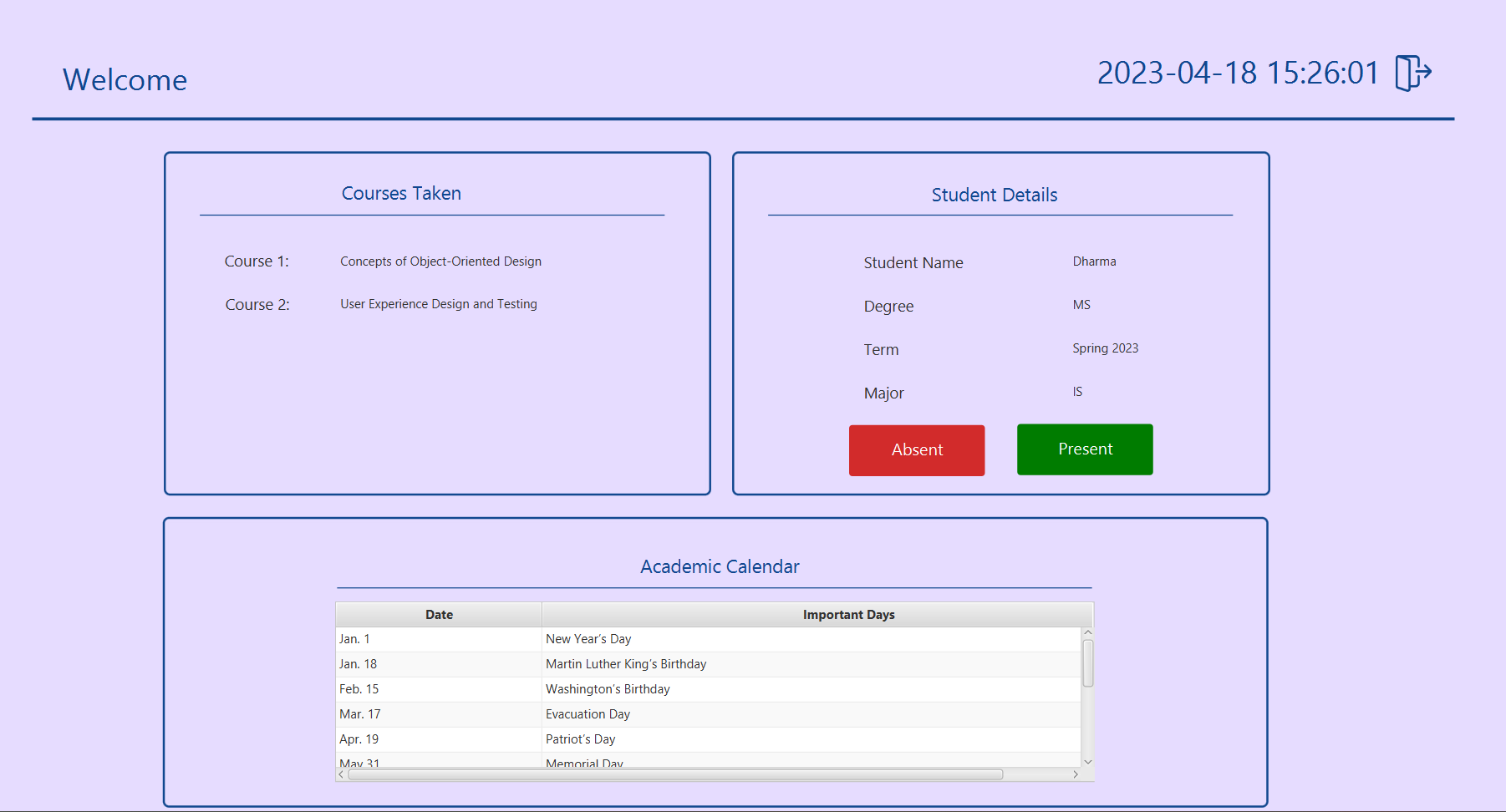
This is the subsection to explain the implementation of part 1.  
 **Login Controller:** The controller includes a method called `userLogIn()` that checks the inputted username and password against a set of predefined values. If the username and password match, the controller loads a new FXML file and sets it as the scene for the current window. The `FXMLLoader` is used to load the FXML file and create a `Parent` object that represents the UI components defined in the FXML file. The `Stage` object is then retrieved from the current button's scene and a new `Scene` object is created using the `Parent` object and set as the scene for the window. If the username and password do not match any of the predefined values, the controller sets the text of a `Label` object to indicate that the login failed.  
**Student Controller:** The controller initializes the view components and defines their behavior when interacting with the user. The class loads student data from a data source, sets up a table to display the data, and populates the table with the loaded data. It also provides methods to load different views when the user interacts with the system. Additionally, the class handles actions like logging out and redirecting the user to the home page. The implementation relies on the use of FXML files to define the layout of the user interface and the organization of the view components.  
**NewStudentController:** This is a JavaFX controller class named `NewStudentController`. It contains methods to handle the actions of the buttons `add` and `cancel`. When the `add` button is clicked, the `addStudent` method of the `StudentData` class is called to add a new student with the data entered in the text fields and combo boxes. A success message is displayed in an `Alert` window, and the `Stage` is closed. When the `cancel` button is clicked, a message is displayed in an `Alert` window, and the `Stage` is closed. The `initialize` method loads data into the `ComboBox`es `course1` and `course2` from a file named "CourseData.txt". The `loadComboBox` method reads the file and adds the first element of each line as an item to the combo box. The `loadStage` method loads the FXML file of a given screen and sets it as the child of the `AnchorPane` named `newStudentPane`.  
**FacultyController:** It implements the Initializable interface and includes several FXML elements such as TableView, TableColumn, Button, and ImageView. The controller loads data from a FacultyData class and populates the TableView with it. It also includes methods to handle events such as logging out, redirecting to the home view, and opening a new stage for adding a new faculty member. The loadStage method loads an FXML file and sets it as the content of the facultyPane, while the loadNewStage method creates a new stage and loads an FXML file to display it. The implementation uses JavaFX scene graph API to build the user interface and the JavaFX event handling mechanism to handle user input.  
**Course Controller:** It implements the Initializable interface and defines methods to handle button clicks and load new stages. The initialize method loads the list of courses using a CourseData object and displays each course in a TitledPane within an Accordion control. The handleButtonClicks method determines which button was clicked and opens the appropriate view. The loadNewStage method loads a new view and displays it in a new window. The loadStage method loads a view and displays it in the same window. The logOut and redirectHome methods redirect to the login and home views, respectively. Overall, the class provides functionality for displaying a list of courses and navigating to other views.  
**NewCourseController:** The controller handles the user interactions with the GUI and communicates with the model (CourseData) to add new courses to the database. The GUI includes text fields for the course title, description, department, and code, as well as a drop-down menu for the faculty. The controller loads the faculty data from a text file and populates the drop-down menu accordingly. When the user clicks the "Add" button, the controller retrieves the information from the text fields and the drop-down menu and sends it to the CourseData model to add the new course. If the user clicks the "Cancel" button, the controller simply closes the GUI window. Alert messages are displayed to the user to inform them if the course was added successfully or not.  
**Controller:** The `Controller` class is responsible for handling the main menu and loading the corresponding FXML files. It implements the `Initializable` interface, which requires the `initialize()` method to be implemented. The `handleButtonClicks()` method listens for button clicks on the main menu and loads the corresponding FXML file using the `loadStage()` method. The `loadStage()` method loads the FXML file and replaces the contents of the `rootPane` with the loaded FXML file. The `logOut()` method loads the Login screen when the user logs out. The FXML files that are loaded include Dashboard, Student, Courses, Faculty, Departments, and Announcements. The `FXMLLoader` is used to load the FXML files, and the `AnchorPane` is used to display the loaded FXML files.  
**UserDashborad:** This is an implementation of a JavaFX user dashboard application. The dashboard displays a clock, calendar, and user information, such as name, degree, major, and enrolled courses. The application is initialized with data loaded from external files. The user can also log out and be redirected to the login screen. The implementation uses JavaFX components such as TableView, TitledPane, Label, and Button. It also utilizes the Model-View-Controller (MVC) design pattern, where the controller class handles user input, updates the model, and updates the view accordingly.

***B.Homefolder:*Main:** This Java program implements a JavaFX application that loads the user interface from an FXML file called "Login.fxml" and displays it on the screen. The program is called "School Management System" and it is designed to provide a graphical user interface for managing school-related tasks.   
The program creates a new instance of the JavaFX Stage class, which represents the main window of the application. The FXML file is loaded using the FXMLLoader class, which is a utility class provided by the JavaFX library for loading FXML files. The loaded FXML file is used to create a Parent object that represents the root node of the scene graph. The Stage object is configured with a title using the setTitle() method. A new Scene object is created and set as the scene for the Stage using the setScene() method. The root node of the scene graph is set as the root of the Scene object. Finally, the Stage is displayed on the screen using the show() method. The program also includes a main() method, which is the entry point of the application. It simply calls the static launch() method of the Application class, which starts the JavaFX runtime and initializes the application.  
  
**C**.***Datafolder:*CourseData:** This Java code provides methods to load and add data to a collection of courses represented by `CourseModel` objects. The data is read from a file named "CourseData.txt" and stored in the `coursesList` collection. The `loadData()` method uses the `Files` and `Paths` classes from the `java.nio.file` package to read all lines of the file, map each line to a `CourseModel` object, and collect them into a list. The `addCourse()` method writes a new course entry to the same file, using the `Files` and `Paths` classes to append the entry to the end of the file. It also adds a new `CourseModel` object to the `coursesList` collection. Both methods throw an `IOException` if an error occurs during file I/O.  
**DashboardData:** The `DashboardData` class contains static fields and methods that are responsible for loading and processing data related to the dashboard of the School Management System. The `loadData()` method reads the student data from the `StudentData.txt` file and processes it to generate statistics that are displayed in the dashboard. It creates two hash maps, `studentByDepartment` and `studentByCourse`, to keep track of the number of students enrolled in each department and course. It then populates an `XYChart.Series` object with the department-wise student count and an `ObservableList<PieChart.Data>` object with the course-wise student count. The `dashboardData` field is an `ObservableList` of `DashboardModel` objects that is used to store the dashboard data. This list is not populated by the `loadData()` method but is used to display the data in the UI. Overall, the `DashboardData` class demonstrates the use of JavaFX charting API to generate statistical data and how to populate data in observable lists that can be used for display purposes in the UI.  
**Faculty Data:** The `FacultyData` class loads and manages the data related to faculties. The `loadData()` method reads the data from a file named `FacultyData.txt` and populates the `facultyData` observable list. The file contains comma-separated values for each faculty member, including their ID, name, department, years of experience, and area of interest. The method uses `Files.readAllLines()` to read all lines from the file, then uses the `map()` method of a stream to split each line into details and create a `FacultyModel` object for each. The `collect()` method is used to collect the resulting objects into a list. Finally, the `FXCollections.observableArrayList()` method is used to create an observable list from the list of faculty models, which is assigned to the `facultyData` field.  
**StudentData:** The `StudentData` class is responsible for loading and updating the data related to the students. The class has a static `idGenerator` field that is used to generate unique IDs for new student entries. The `loadData` method reads the data from a file named "StudentData.txt" and populates an `ObservableList` of `StudentModel` objects. The method also updates the `idGenerator` by reading the last line of the file and incrementing the ID by 1.  
The `addStudent` method adds a new student to the file and the `ObservableList` of `StudentModel` objects. The method generates a new ID using the `idGenerator` and appends the new student entry to the file using the `Files.write` method. The new student object is then added to the `ObservableList`.Overall, the `StudentData` class provides a way to load and update student data using a file-based system.

# **V. Evaluation**

The Student Management System has been implemented successfully, and a sample run of the system shows that it meets the requirements of the problem statement. The system has a user-friendly interface with easy navigation to the different modules.  
**Figure 5** shows login page where user can login as a Admin or a student if the profile is already created by admin.

…  
  
Figure 5. Login Page

  
Figure 6. Admin Dashboard  
  
  
Figure 7. Statistics Page  
  
  
Figure 8. Faculty Page  
  
  
Figure 9. Departments Page  
  
  
Figure 10. Students page  
  
  
Figure 11. Courses Page  
  
  
Figure 12. Students Dashboard

**Figure 6** shows the admin page of the system, which provides access to the different modules. The admin can navigate to the statistics page, faculty page, departments page, courses page, and students page using the buttons on the page.   
**Figure 7** shows the statistics page provides visualizations of key performance indicators. This page is showing the statistical information regarding the student capacity per department. We can observe that most popular departments are CSYE, INFO, IS and the least popular are DAMG, DA and Computer Science.

Each course in a particular department is assigned a different color which is visible on the legend. The pie chart represents the volume of students present in each course.  
**Figure 8 and 9** faculty page and departments page allow for management of instructors and   
departments, respectively.  
**Figure 10** the students page allows for the creation and management of student records. We can add new student as shown in the images, user will be able to choose his course 1 and course 2 as shown in the image.  
**Figure 11** The courses page allows for the creation and management of courses.  
**Figure 12** Once user login as a student he’ll be navigated to the student’s dashboard as shown in the image. Once logged in, students can view their personal details. They can also view the courses they have taken, and record their attendance.  
Additionally, students can also view the academic calendar that provides students with important dates and events.  
This helps students to stay informed and plan their studies accordingly.

# **VII. Discussion (Reflection)**

**The Idea:**

The Student Management System was designed to tackle the challenges faced by educational institutions in managing student records. The aim was to create a centralized system that would enable teachers and administrators to manage all aspects of student records, from registration to course management and statistics, in one place. The system was designed to be user-friendly, efficient, and accessible from any device with an internet connection.

**The Challenges:**

One of the major challenges faced during the development of the Student Management System was ensuring that the system was scalable and could handle large amounts of data as well as data movement properly. The system had to be able to manage the records of thousands of students, faculty members, and courses without slowing down or crashing. Another challenge was designing an interface that was easy to use and navigate for both students and administrators, even those with limited technical knowledge.  
One of the main hurdles this project encountered is the intuitiveness of the User Interface. As it is the facade of the application, it should be as modern as possible without sacrificing the fluidity of the application. It took countless hours of research, which ended up in learning of uncharted approaches of JavaFX.  
  
**The Outcome:**

Apart from learning and exploring the concepts like

- JavaFX  
- Class Definition  
- Inheritance/Polymorphism  
- Abstract Classes/Interfaces  
- Iterators  
- Lists

We also gained knowledge about Java's file handling APIs.  
and tools like  
-Git repository  
-Git GUI  
-Scene Builder  
-Draw.io  
-Figma

# **VIII. Conclusions and Future Work**

Please summarize the findings of the project; You can also try to answer any of following questions:

* What are the advantages or benefits of using your solution?
* What are the problems found but not yet explored in the project?
* If your team has more time, what do you want to improve?

**In conclusion**, the Student Management System is an effective solution that simplifies the management of student records in educational institutions. The system provides a user-friendly interface that allows teachers and administrators to store, manage, and retrieve student records easily. Through the implementation of modules for student registration, course management, and course statistics, the system enables administrators to efficiently track student progress and performance.

**The advantages of using this solution** include the automation of the entire student record-keeping process, which saves time and effort for teachers and administrators. The system also provides a centralized database for all student information, which ensures data accuracy and consistency. Additionally, the system allows for easy access to information, which facilitates communication between teachers, administrators, and students.

**One of the challenges** encountered during the project was the design of the user interface, which required extensive planning and testing to ensure ease of use and navigability. However, the final system design was successful in addressing this challenge.

**If we had more time**, we would focus on improving the system's security features, such as implementing user authentication and authorization protocols to ensure data confidentiality and integrity. We would also integrate a messaging system that would enable seamless communication between students, teachers, and administrators.

**In future work**, we suggest exploring the integration of machine learning algorithms to analyze student performance data and generate insights that could aid in decision-making for educators. Additionally, the system could be expanded to include features for tracking student attendance and generating reports on student performance.

# **IX. Job Assignment**

* *Srinivasa Pavan Ganesh Vasa* – Courses Page, Students Page, Statistics Page on Admin side, System Architecture and diagram.
* *Dharma Thanishq Nimmala* – Login Page, Student Dashboard, Admin Dashboard, Power Point presentation, layout design.
* *Rushikesh Karwankar* – UI design (Figma), Departments, Faculty dashboard on Admin side, workflow and diagram.

**X. References**

1. Althobaiti, Maha & Mayhew, Pam. (2016). Assessing the Usability of Learning Management System: User Experience Study. 9-18. 10.1007/978-3-319-28883-3\_2..
2. Almaiah, M.A., Al-Khasawneh, A. & Althunibat, A. Exploring the critical challenges and factors influencing the E-learning system usage during COVID-19 pandemic. Educ Inf Technol 25, 5261–5280 (2020). https://doi.org/10.1007/s10639-020-10219-y Omar A. Al-Mohrej and Nader S. Al-Kenani. Acute ankle sprain: conservative or surgical approach? *EFORT Open Rev.* 2016 Feb; 1(2): 34–44.
3. Singh R, Singh R, Kaur H, Gupta O. P. Development of Online Student Course Registration System. Orient.J. Comp. Sci. and Technol;9(2)
4. Kuppusamy, Palanivel. (2020). SERVICE-ORIENTED REFERENCE ARCHITECTURE TO SMART EDUCATION.
5. https://www.geeksforgeeks.org/classes-objects-java/.
6. https://www.codecademy.com/learn/learn-java/modules/learn-java-inheritance-and-polymorphism/cheatsheet.
7. https://www.geeksforgeeks.org/interfaces-in-java/.
8. https://www.geeksforgeeks.org/iterator-vs-collection-in-java/
9. https://www.geeksforgeeks.org/difference-between-list-and-set-in-java/
10. https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/Button.html