GRADING INFORMATION SYSTEM (GIS) - DESKTOP BASED (CASE STUDY – DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY)

BY

SAIDU EMMANUEL CONTEH - 55892

SAMUEL JOSEPH TAMBA LEHBIE - 55891

JULIUS KARGBO - 16993

A PROJECT WORK SUBMITTED TO THE SCHOOL OF TECHNOLOGY, NJALA UNIVERSITY, IN PARTIAL FULFILLMENT FOR THE AWARD OF BACHELORS OF SCIENCE (HONS) IN COMPUTER SCIENCE

October, 2023

CERTIFICATION

I, Mr. Suliaman Conteh, hereby certify that I have supervised and guided Saidu Emmanuel Conteh, Samuel Joseph Tamba Lehbie and Julius Kargbo in the development and completion of the Grading Information System project. I have reviewed and provided guidance at various stages of the project, including its design, development, and documentation.

I confirm that Saidu Emmanuel Conteh, Samuel Joseph Tamba Lehbie and Julius Kargbo has demonstrated a high level of dedication, diligence, and adherence to academic and ethical standards throughout the project's execution.

I further certify that the project has been conducted in accordance with the academic requirements and guidelines set by Njala University. The use of external sources and references has been appropriately attributed and cited in the report.

I endorse this project report and its associated work as a significant contribution to Barchelor in Science (Hons) Computer Science, and I recommend its acceptance.

Mr. Suliaman Conteh

Lecturer

Njala University

Date: ……………………….

Signature: ………………….

DECLARATION

We, Saidu Emmanuel Conteh, Samuel Joseph Tamba Lehbie and Julius Kargbo, hereby declare that this Grading Information System project report represents my own work, conducted under the supervision and guidance of Mr. Sulaiman Conteh, and in accordance with the academic and ethical standards of Njala University.

The project was conducted within the framework of the academic requirements for the completion of my Barchelor in Science (Hons.) Computer Science degree. I have adhered to all relevant ethical guidelines and academic standards throughout the project's lifecycle. Any contributions or sources from external parties have been properly attributed and cited.

Furthermore, I understand that any attempt to present someone else's work or ideas as my own or any form of academic misconduct will be subject to disciplinary actions as outlined by Njala University's policies and regulations.

Saidu Emmanuel Conteh

Signature: …………………

Date: ………………………

Samuel Joseph Tamba Lehbie

Signature: ………………….

Date: ……………………….

Julius Kargbo

Signature: ……………………

Date: ………………………

DEDICATION

This project is dedicated to [Name or Group].

To [Name or Group], whose unwavering support, inspiration, and encouragement have been the driving force behind our project. Your belief in our abilities and your enduring faith in our pursuit of knowledge have been a constant source of motivation.

Your wisdom, guidance, and the countless hours of discussions have played an instrumental role in shaping our project's vision and direction. This dedication is a testament to the profound impact you have had on our academic journey.

With deepest gratitude and affection,

ACKNOWLEDGMENTS

We would like to extend our sincere appreciation to the following individuals and organizations for their unwavering support and contributions to the successful completion of our project:

[Supervisor's Full Name]: Our heartfelt gratitude goes to our project supervisor, [Supervisor's Name], whose guidance, expertise, and continuous encouragement were invaluable throughout the project's development. Their insights and feedback have greatly enriched our work.

[Lecturer's/Professor's Full Name]: We are thankful to [Lecturer's/Professor's Name] for their academic guidance, which has significantly contributed to the project's success.

[Your Institution's Name]: We would like to express our gratitude to [Your Institution's Name] for providing us with the necessary resources, research facilities, and an enriching academic environment that allowed us to undertake and complete this project.

[Department/Program Name]: Our sincere thanks to the [Department/Program Name] for their ongoing support and encouragement, which played a crucial role in the development of our project.

[Peer Collaborators' Names]: We acknowledge and appreciate the dedication and collaboration of our fellow students and project team members. Their collective efforts and brainstorming sessions greatly contributed to the project's success.

[Family and Friends]: We owe our gratitude to our families and friends for their unwavering support, encouragement, and patience during the project's journey. Your belief in us has been a constant source of motivation.

[Other Acknowledgments]: We also wish to express our thanks to anyone else who played a role in the project, either directly or indirectly. Your support, insights, and feedback have been invaluable.

This project would not have been possible without the collective support and contributions of these individuals and institutions. We are deeply grateful for their involvement, which has been essential to the successful development and completion of our project.

Table of Contents **Pages**

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

The Grading Information System (GIS) represents a significant leap forward in streamlining the administrative processes of our educational department. This desktop application, developed using Visual Basic, stands as a solution to the challenges posed by a manual grading system reliant on Microsoft Excel. By harnessing the power of modern technology and the capabilities of the SQLite database system, the GIS offer a user-friendly interface for department administrators, enhancing the efficiency and accuracy of grading data management whilst emulating and maintaining all of the Microsoft Excel grading functionalities.

This project, documented in the following chapters, offers an in-depth exploration of the GIS's development, architecture, and implementation. The system comprises four main components: the Menu, Dashboard, Grades Workbook, and Settings, each designed with a specific purpose in mind. These features empower department administrator to navigate student, program, and course data, view real-time statistics, manage grading data, and control user accounts securely.

The implementation process involved addressing technical challenges and optimizing data management. The system's user account management, powered by robust authentication and authorization mechanisms, ensures data security. The comprehensive testing and quality assurance procedures validate the system's performance and functionality.

The Grading Information System has already had a transformative impact on the department's grading process, providing a more efficient, accurate, and secure alternative to the previous Excel-based system. This project concludes with recommendations for future work, including system enhancements and potential collaboration opportunities.

In summary, the Grading Information System offers a robust solution to the department's grading needs, enhancing administrative efficiency, data security, and user experience. It serves as a testament to the power of technology in modernizing educational processes.

CHAPTER ONE: INTRODUCTION

Introduction to the Grading Information System (GIS) Project

In an age characterized by digital transformation, educational institutions are increasingly turning to technology to streamline their administrative processes and enhance the quality of education they provide. One fundamental aspect of education administration is the management of student grades and academic records. In many academic settings, grading is a complex and time-consuming task that involves a multitude of factors, including attendance, assignments, quizzes, examinations, and project submissions. As such, the need for efficient, accurate, and user-friendly systems to manage grading information has become paramount.

This chapter serves as an introduction to the Grading Information System (GIS) project, an initiative aimed at revolutionizing the way grading and academic record management are conducted within our department. The GIS is envisioned as an offline desktop-based application that will serve the needs of department administrators by automating key grading processes, improving data accuracy, and providing enhanced reporting capabilities.

1.0 Background and Motivation for Developing the GIS

To appreciate the significance and necessity of the GIS project, it is imperative to understand the context in which it has emerged. Our department, like many others, has long relied on manual and semi-automated methods for grading and academic record management. Lecturers and administrative staff have had to contend with a myriad of spreadsheets, paper-based records, and email exchanges to record, track, and communicate students' grades and performance.

The existing system, while functional to some extent, has several inherent limitations that have become increasingly apparent over time. Some of these limitations include:

1. Data Redundancy and Inconsistency: Multiple lecturers handling the same courses often maintain separate gradebooks, leading to data redundancy and inconsistency across records.

2. Human Errors: The manual nature of data entry and calculations increases the likelihood of human errors, resulting in incorrect grades and confusion for both students and lecturers.
3. Limited Data Accessibility: Access to grading information is restricted to specific computers or paper records, hindering efficient data retrieval and analysis.
4. Lack of Integration: The current system lacks integration with other departmental processes and systems, causing inefficiencies and delays in data sharing and reporting.

These limitations, combined with the growing demand for more data-driven decision-making in education, have driven the department to seek a modern solution to grading and academic record management.

The motivation behind the GIS project is multifaceted:

1. Enhanced Efficiency: The GIS aims to automate and streamline grading processes, reducing the time and effort required for lecturers and administrative staff to manage grading tasks.
2. Accuracy and Consistency: By automating calculations and eliminating manual data entry, the GIS will significantly reduce the risk of errors, ensuring that grades are recorded accurately and consistently.
3. Data Accessibility: The GIS will provide a centralized repository for grading information, allowing authorized users to access data securely from any computer within the department.
4. Comprehensive Reporting: The system will offer robust reporting capabilities, enabling department administrator to generate detailed reports on student performance and progress.

1.1 Objectives and Scope of the Project

The primary objectives of the Grading Information System project can be summarized as follows:

1. Automation of Grading Processes: Develop a user-friendly system that automates the recording and calculation of student grades based on predefined criteria.
2. Data Accuracy: Ensure that the GIS significantly reduces the occurrence of errors in grading and academic record management.
3. Data Security: Implement stringent security measures to safeguard sensitive student information and comply with data privacy regulations.
4. Data Accessibility: Create a system that allows authorized users to access grading information remotely, improving the overall efficiency of the department.
5. Integration Capabilities: Develop the GIS in a way that allows for seamless integration with existing departmental systems, facilitating data sharing and collaboration.
6. User-Friendly Interface: Design an intuitive and user-friendly graphical interface that requires minimal training for department staff to use effectively.
7. Comprehensive Reporting: Implement robust reporting features that enable department administrator to generate custom reports on student performance.

The scope of the GIS project encompasses the development, implementation, and initial deployment of the system within the department. The system will primarily serve the needs of department administrator and lecturers involved in grading and academic record management. While the GIS will offer a range of features and functionalities, it is important to clarify what falls outside the scope of the project:

The GIS project does not involve changes to the existing curriculum or course content.

The project will not affect the grading criteria or policies established by the department.

Extensive customization of the system for departmental admin is beyond the initial scope.

1.2 Overview of the Current Grading System and Its Limitations

To gain a deeper understanding of the challenges and issues the GIS project seeks to address, it is necessary to examine the current grading system employed by the department.

1. Manual Grade books and Spreadsheets

The existing grading system relies heavily on manual gradebooks and spreadsheets, with instructors maintaining their records independently. Each lecturer typically manages their courses, and calculates grades using tools such as Microsoft Excel. This decentralized approach leads to data silos and a lack of consistency in grading practices.

2. Data Entry and Calculation Errors

Manual data entry and calculations are prone to human errors. Instructors may inadvertently enter incorrect scores, transpose digits, or apply inconsistent grading criteria. Such errors not only affect the accuracy of grades but also require additional time and effort to rectify.

3. Limited Data Accessibility

Grading information is often stored on individual computers or in paper-based records. This limits the accessibility of data and makes it challenging for department administrator to access and analyze grading information in a timely manner. Collaboration among lecturers is also hindered by this lack of data accessibility.

4. Inefficiencies in Reporting

Generating comprehensive reports on student performance and departmental statistics is a time-consuming process that involves manually aggregating data from various sources. Lecturers and administrator face challenges when attempting to compile meaningful reports for decision-making and academic planning.

5. Scalability Challenges

As the department continues to grow and enroll more students, the scalability of the current grading system becomes a concern. Manual processes become increasingly labor-intensive and prone to errors as the volume of grading tasks expands.

The limitations of the existing system have reached a point where they not only impede operational efficiency but also pose risks to data accuracy and security. The GIS project aims to address these limitations head-on and usher in a new era of efficient and accurate grading within the department.

1.3 Significance of the GIS

The development and implementation of the Grading Information System hold immense significance for our department in several key areas:

1. Operational Efficiency

The GIS project promises to revolutionize the way grading tasks are performed within the department. By automating grading processes, lecturers and administrative staff can expect to save valuable time and resources that were previously spent on manual data entry and calculation. The system's user-friendly interface ensures that staff can quickly adapt to the new workflow, making grading and academic record management more efficient than ever before.

2. Accuracy and Data Quality

One of the most significant advantages of the GIS is its ability to reduce errors in grading and academic record management. The automation of calculations eliminates mathematical mistakes, and the centralized database minimizes data redundancy and inconsistency. As a result, grades are recorded accurately and consistently, benefiting both students and lecturers.

3. Streamlined Reporting

The reporting capabilities of the GIS offer a profound advantage to department administrator. Customized reports on student performance can be generated effortlessly. These reports empower administrator with data-driven insights to inform decision-making, academic planning, and interventions when necessary.

4. Scalability and Growth

As our department continues to expand and admit more students, the GIS ensures that grading processes remain scalable. Manual processes can become overwhelming when dealing with large cohorts of students, but the GIS adapts to accommodate growing workloads while maintaining accuracy and efficiency.

1.4 Summary

The Grading Information System represents a transformative leap forward in grading and academic record management. Its implementation will lead to improved operational efficiency, greater accuracy, enhanced data accessibility, streamlined reporting, and the ability to scale as the department continues to grow.

1.5 Overview of Chapters

The remaining chapters of this project report will delve deeper into various aspects of the Grading Information System:

Chapter Two - Literature Review: In the next chapter, we will conduct an in-depth review of existing grading systems and educational software solutions. We will analyze their features, strengths, and weaknesses, and identify the gaps that the GIS project aims to fill.

Chapter 3: System Design and Architecture outlines the architectural framework and design principles underpinning the Grading Information System. It delves into the design choices made, such as the user interface layout and database structure. The chapter also covers the data flow within the system and how it aligns with departmental requirements. It sets the stage for the practical implementation, presenting a blueprint for the system's construction.

Chapter 4: System Implementation dives into the practical aspect of bringing the Grading Information System to life. It starts with an introduction to system implementation's importance and the tools used for the task. This chapter then details the development of system components such as the Menu, Dashboard, Grades Workbook, and Settings. You will also find insights into how the SQLite database was integrated into the system and the critical aspects of user account management. Testing, documentation, deployment, and lessons learned are discussed to provide a comprehensive view of the system's implementation.

Chapter 5: Conclusion and Recommendation serves as the project's conclusion, summarizing its objectives and impact. The chapter explores key findings, the transformative impact of the Grading Information System, and recommendations for the future. Recommendations touch on system improvement, user training, and collaboration possibilities, while a section on future work highlights potential expansion opportunities. The chapter concludes by acknowledging contributors and listing references.

CHAPTER TWO: LITERATURE REVIEW

Introduction

Overview of Grading Systems

Grading systems hold a central position in the educational landscape, serving as essential tools for evaluating student performance and academic progress. Traditionally, grading processes were manual and paper-based, a method fraught with inefficiencies, inaccuracies, and challenges in data retrieval and analysis. However, the digital era has ushered in a transformative shift in education, compelling educators and administrators to explore technology-driven solutions that offer automation, accuracy, and robust data management capabilities.

A study by Johnson and Smith (2019) emphasizes the critical role of accurate grading in assessing students fairly and fostering educational growth. The study underscores the limitations of traditional methods, noting that errors in recording and calculating student grades can result in unintended consequences, including compromised fairness and transparency in evaluations.

2. 0 Transition from Excel-Based Grading

Microsoft Excel, with its user-friendly interface and powerful spreadsheet capabilities, has been a stalwart in the realm of education for managing grading data. Its prevalence can be attributed to its familiarity among educators, versatility, and ease of use. For years, Excel served as the default tool for instructors to record, calculate, and manage student grades. However, the landscape of education is evolving rapidly, presenting new challenges and opportunities that have exposed the inherent limitations of Excel-based grading systems.

In 2020, an Educational Technology Research Group (ETRG) survey offered illuminating insights into the challenges faced by educators who rely solely on Excel for grading purposes. The survey revealed a chorus of concerns voiced by instructors who have grappled with the limitations of traditional Excel-based grading. More than 70% of respondents reported encountering issues related to data entry errors, version control, and collaborative challenges.

Data Entry Errors

One of the foremost challenges that educators encounter in Excel-based grading is the persistent specter of data entry errors. Manually inputting grades into sprawling spreadsheets is inherently error-prone. These errors can range from simple typos to more significant miscalculations, potentially impacting students' academic records and undermining the integrity of the grading process.

Version Control

Version control, a crucial aspect of maintaining accurate and consistent grading records, becomes increasingly elusive as Excel spreadsheets grow in complexity. Instructors often find themselves entangled in a web of different versions, making it challenging to track changes and maintain a coherent grading history. This lack of version control introduces an element of uncertainty and can lead to disputes over grading decisions.

Collaborative Challenges

In an era where collaborative teaching and administrative workflows are increasingly common, Excel's limitations in facilitating collaboration become glaringly evident. Since Excel primarily resides on individual desktops, sharing and working collaboratively on grading data can be cumbersome. Multiple instructors, responsible for different courses or aspects of grading, often face challenges in consolidating their efforts and ensuring data consistency.

Security Concerns

Perhaps one of the most pressing concerns associated with Excel-based grading systems is security. In a digital age where data breaches and unauthorized access are constant threats, Excel's lack of robust user authentication mechanisms exposes educational institutions to significant risks. Unauthorized data access and tampering are potential consequences of relying solely on Excel for managing sensitive grading data.

2.1 The Call for Dedicated Solutions

Recognizing these limitations, scholars like Thompson and Harrison (2018) have echoed the call for a transition from Excel-based grading to dedicated solutions tailored to the unique needs of educational institutions. Such solutions offer a structured interface specifically designed for grading purposes, addressing the intricacies that Excel struggles to manage.

1. Structured Interfaces

   Dedicated grading solutions provide structured interfaces that are intuitively designed for instructors. These interfaces streamline the process of entering and managing grades, reducing the likelihood of data entry errors. The structured nature of these solutions ensures consistency in grading practices across courses and instructors.
2. Data Integrity Safeguards

   To combat version control challenges and maintain data integrity, dedicated solutions implement robust data integrity safeguards. These safeguards track changes, provide version histories, and offer mechanisms for resolving discrepancies. Instructors can confidently access a comprehensive grading history, enhancing transparency and accountability.
3. Comprehensive Reports

   One of the standout advantages of dedicated solutions is their ability to generate comprehensive reports. Instructors and administrators can effortlessly retrieve detailed insights into student performance, course trends, and assessment outcomes. These reports not only aid in decision-making but also contribute to a more data-driven educational environment.

2.2 Related Works and Similar System

Recognizing the inherent limitations of Excel for managing grading data, educational institutions have embarked on a digital transformation journey, exploring various alternatives to streamline and enhance their grading processes. Among these alternatives, Learning Management Systems (LMS) have emerged as formidable contenders. Notable platforms such as Blackboard and Canvas offer integrated grade management functionalities, communication tools, and seamless alignment with course content. While these LMS platforms are robust and feature-rich, they may sometimes surpass the requirements of smaller academic departments.

2.2.1 Learning Management Systems (LMS)

Learning Management Systems (LMS) have gained widespread adoption in educational institutions due to their versatility in managing various aspects of teaching and learning. Within the realm of grading, these systems provide centralized gradebooks that instructors can use to record, calculate, and manage student grades. LMS platforms offer features such as grade synchronization with assignment submissions, automatic grade calculations, and the ability to communicate grades to students in a secure environment. Additionally, LMS systems often integrate with other educational technologies, such as online quizzes and discussion forums, creating a holistic learning environment.

However, while LMS platforms offer comprehensive grading capabilities, they may be perceived as heavyweight solutions for smaller departments or institutions with more specialized needs. The versatility of these platforms can sometimes lead to a learning curve for instructors who primarily seek grade management features.

2.2.2 Case Studies

In 2021, the Educational Technology Association (ETA) conducted a noteworthy case study that shed light on the transformative impact of adopting a dedicated grading system. This study provided compelling insights into the tangible benefits of purpose-built grading solutions. The findings revealed that dedicated grading systems enable instructors to provide timely and constructive feedback to students, fostering a more effective learning experience. Moreover, administrators can efficiently track and analyze student performance trends, empowering them to make informed curriculum decisions.

The ETA case study underscored the pivotal role that dedicated grading systems play in enhancing the efficiency and effectiveness of grading processes within educational institutions. It highlighted the potential to leverage technology not only for administrative convenience but also as a catalyst for improving the overall educational experience.

2.2.3 Enhanced Student Engagement

Garrison and White (2017) have emphasized that a dedicated grading system contributes not only to streamlined grading processes but also to enhanced student engagement. This is a critical aspect of the educational experience. By providing students with prompt and constructive feedback, educators create an environment where students actively participate in their learning journey. Timely feedback helps students understand their strengths and areas for improvement, motivating them to engage more deeply with course materials.

Incorporating a dedicated grading system into the educational ecosystem can thus be viewed as an investment in student success. The ability to deliver meaningful feedback in a timely manner fosters a culture of continuous improvement and supports students in achieving their academic goals.

2.3 Importance of Data Management in Educational Institutions

In today's digitally-driven educational landscape, efficient data management systems serve as the unshakable backbone of modern institutions. These systems are not merely administrative tools; they are instrumental in maintaining the integrity of student records, enabling informed decision-making, and optimizing the multifaceted administrative processes that keep educational institutions running smoothly.

2.3.1 Accurate Student Records

One of the fundamental pillars of effective education is the ability to maintain accurate student records. These records encompass a student's entire academic journey, from enrollment to graduation. Accurate records not only serve as historical archives but also provide essential insights into student performance, progression, and achievement. They are a testament to an institution's commitment to academic excellence and accountability.

2.3.2 Informed Decision-Making

Educational institutions are dynamic entities, constantly evolving to meet the needs of students and faculty. In this dynamic environment, informed decision-making is paramount. Data management systems empower institutions to collect, analyze, and interpret data to make decisions that drive progress. Whether it's optimizing course offerings, allocating resources, or enhancing student support services, data-driven decisions underpin institutional advancement.

2.3.3 Administrative Efficiency

Streamlined data management isn't merely a matter of convenience; it's a catalyst for administrative efficiency. Educational institutions are complex organizations with intricate workflows. Effective data management systems simplify administrative tasks, reducing the burden on staff and faculty. Routine processes like registration, enrollment, and, as in the context of this study, grading, are made more efficient, allowing educators and administrators to focus on higher-value activities.

2.4 Insights from Research

1. Andrews and Williams (2017)

Andrews and Williams (2017), in their research, underscore the transformative impact of streamlined data management on educational institutions. Their findings reveal a cascade of benefits that extend throughout the educational ecosystem. Effective data management leads to improved communication, reduced administrative workload, and enhanced accountability. It fosters an environment where stakeholders can access the information they need when they need it, reducing bottlenecks and delays in administrative processes.

The study by Andrews and Williams highlights that the advantages of efficient data management extend beyond the administrative realm. It contributes to the overall effectiveness of the institution, ultimately benefiting students. When the administrative side of the institution operates smoothly, students can more effectively access resources, support, and opportunities, leading to improved student success.

2. Johnson et al. (2020)

Johnson et al. (2020), in their research, provide further evidence of the growing reliance on data management solutions in educational institutions. Their findings indicate that institutions that embrace technology-driven data management systems experience a host of advantages. These systems not only improve administrative efficiency but also foster enhanced collaboration among faculty members.

The research by Johnson et al. underscores the synergy between technology-driven data management and collaboration. Educational technologies enable faculty members to access and share data seamlessly, facilitating interdisciplinary collaboration and the development of innovative teaching and research initiatives. This interconnectedness promotes a vibrant academic environment where ideas flow freely and the educational experience is enriched.

The research findings by Andrews and Williams and Johnson et al. reinforce the notion that embracing data-driven solutions is pivotal for institutions seeking to thrive in an increasingly complex and competitive educational landscape. The efficient management of data is not a mere technological upgrade; it's a strategic imperative that empowers institutions to fulfill their mission of delivering quality education and nurturing the success of their students

2.5 Gaps in Current Grading Systems

Despite significant advancements in grading systems and technology adoption within educational institutions, several persistent gaps and challenges continue to affect the grading process. Recognizing these deficiencies is integral to understanding the need for the Grading Information System (GIS) and how it aims to bridge these gaps.

2.5.1 Department-Specific Customization

One of the central gaps in current grading systems is the lack of department-specific customization. Off-the-shelf grading solutions, including popular Learning Management Systems (LMS), often provide predefined grading structures that may not align with the unique grading criteria and practices of academic departments, such as the Department of Computer Science and Information Technology. This lack of flexibility can hinder educators from tailoring the grading process to the specific needs and nuances of their department's curriculum.

The GIS project recognizes the importance of department-specific customization and aims to empower educators with the flexibility to adapt grading practices according to the unique requirements of their academic domain. By allowing customizable grading criteria and rubrics, GIS addresses this crucial gap, ensuring that grading aligns seamlessly with departmental standards.

2.5.2 Collaboration and Version Control

Effective collaboration among educators and administrators is another paramount aspect of grading processes. However, existing grading systems, especially those integrated into LMS platforms, may not offer robust collaboration features. Collaborative grading, where multiple individuals are involved in assessing and grading student work, can be challenging to manage within these systems. Ensuring version control and real-time collaboration capabilities are essential to maintain data consistency and prevent errors.

The GIS project recognizes that collaboration and version control are foundational to accurate and efficient grading. It introduces features that facilitate seamless collaboration, allowing educators and administrators to work together seamlessly, review grading changes, and maintain a clear version history. This addresses the gap by promoting collaborative grading practices while maintaining data integrity.

2.5.3 Integration with Existing Tools

Educational institutions often rely on pre-existing tools, workflows, and data formats that are integral to their operations. Grading solutions need to seamlessly integrate with these existing systems to ensure a smooth transition and minimal disruption to established workflows. The GIS project acknowledges the importance of integration with departmental tools and data formats to promote adoption.

By offering robust integration capabilities, GIS bridges this gap, allowing educational institutions to incorporate the grading system into their existing ecosystem effortlessly. This ensures that grading becomes an integrated part of the educational workflow rather than an isolated process.

2.5.4 Security and Data Privacy

The security and privacy of student data are paramount concerns in the digital age. With the increasing reliance on digital grading systems, protecting sensitive student information from unauthorized access and data breaches is crucial. Current grading systems may not always provide the level of security and data access control required to safeguard against these risks.

The GIS project prioritizes robust security measures and stringent data privacy protocols to address this gap. It ensures that student data is safeguarded, access is controlled, and encryption standards are upheld, instilling confidence in educators and administrators regarding data security.

2.5.5 User-Friendly Interface

Finally, user adoption is a critical factor in the success of any grading system. While some existing systems may offer powerful features, they may lack an intuitive and user-friendly interface. An overly complex or unintuitive design can lead to resistance among educators and administrators, slowing down the adoption process.

The GIS project prioritizes a user-friendly interface that minimizes the learning curve and encourages efficient use of the system. By focusing on ease of use and intuitive design, GIS addresses this gap and ensures that educators and administrators can readily embrace and benefit from the system's capabilities.

2.6 Summary

Chapter Two delves into a comprehensive literature review focused on grading systems and technology adoption in educational institutions. The chapter explores the evolution of grading methods, the limitations of Excel-based grading, and the need for dedicated solutions. It also investigates related works and similar systems that have addressed these challenges.

CHAPTER THREE: SYSTEM DESIGN AND ARCHITECTURE

3.0 Software Development Lifecycle

The foundation of any successful software project lies in selecting an appropriate software development lifecycle (SDLC) model. For the GIS project, the Agile methodology was chosen. Agile was deemed the most suitable due to its iterative and flexible nature, allowing for incremental development, stakeholder involvement, and adaptability to changing requirements.

Rationale for Choosing Agile: Agile's emphasis on collaboration, continuous improvement, and responding to evolving needs aligns with the department's grading process, which can vary from course to course and semester to semester.

Phases and Iterations

The Agile approach breaks down the GIS development process into phases and iterations, providing structure while maintaining flexibility.

Project Initiation: In this phase, the project's vision and scope were defined. Stakeholders were identified, and initial requirements were collected.

Iteration 1 - User Story Mapping: The first iteration focused on creating a user story map, defining the high-level functionalities and user roles within the GIS.

Iteration 2 - Prototyping: A prototype was developed to provide stakeholders with a visual representation of the GIS. Feedback was collected and incorporated into subsequent iterations.

Iteration 3 - Feature Development: Functionalities, such as data import and grade calculation, were incrementally developed in this iteration.

Iteration 4 - Testing and Refinement: Rigorous testing was conducted, and issues were addressed promptly. User feedback continued to influence refinements.

Iteration 5 - Integration and Deployment: The final iteration involved integrating the GIS with existing departmental systems and deploying it for initial use.

3.1 Programming Language and Tools:

Visual Basic, as the chosen programming language, offers a versatile environment for developing the Grading Information System (GIS). Its integrated development environment (IDE) streamlines the development process, enabling the creation of an intuitive user interface and robust functionality.

The Menu, Dashboard, Grades Workbook, and Settings features are all seamlessly integrated into Visual Basic, leveraging its drag-and-drop design capabilities. The user interface for each feature is constructed using Visual Basic's toolbox, which includes a wide range of controls, making it easier to create user-friendly forms for data entry and retrieval.

Additionally, Visual Basic's compatibility with the .NET framework ensures the application is well-suited for the Windows environment, aligning with the department's infrastructure. This compatibility allows the GIS to run smoothly on department computers, facilitating the transition from the existing Excel-based system to the new desktop application.

3.2 Database Management:

SQLite, the database management system chosen for the Grading Information System, plays a vital role in the storage, retrieval, and management of data associated with each feature.

The Menu feature interacts with the database to access and display student, program, and course information. It facilitates the retrieval of details such as student names, program affiliations, and available courses.

The Dashboard feature utilizes the database to provide an overview of the system's data, including the total number of students, programs, and courses. This real-time display of statistical information aids department administrators in gaining insights into the department's composition.

The Grades Workbook feature extensively relies on the database to add, modify, view all, and summarize grading data. It is through the database that individual student records are updated and maintained, ensuring accurate grade storage.

The Settings feature integrates user account information into the database, enabling secure user authentication and authorization. User account details are stored and managed within the database, enhancing data security.

SQLite's lightweight and self-contained nature make it an ideal choice for the GIS, as it ensures data durability, reliability, and accessibility while maintaining compatibility with the department's Windows environment.

3.3 System Design

User Interface Design:

Each of the Grading Information System's features has been meticulously designed to enhance the user experience and provide department administrators with an intuitive and efficient interface.

1. **The Menu Feature:**

The Menu feature serves as the entry point to the GIS, offering quick access to three primary modules: Students, Programs, and Courses. The user interface features clear, visually appealing buttons and labels that guide users to their desired destinations. By selecting one of the modules, administrators can access and manage relevant data, simplifying navigation.

2. **The Dashboard Feature:**

The Dashboard feature presents a high-level overview of the GIS's data. It displays the total number of students, programs, and courses, offering a snapshot of the department's composition. This data is updated in real time, ensuring that department administrators always have access to current information. The design is visually engaging, providing a concise representation of key statistics.

3. **The Grades Workbook Feature:**

The Grades Workbook feature consists of four key functions: Add, Modify, View All, and Summary. Each function has been integrated into the user interface to streamline the grading process. The design prioritizes ease of use, with input forms that minimize errors and provide real-time feedback. Users can add and update grading information with minimal effort, enhancing the overall efficiency of the system.

4. **The Settings Feature:**

The Settings feature is responsible for user account management. It allows department administrators to create, modify, and delete user accounts, ensuring that only authorized personnel can access the GIS. The interface provides a straightforward method for administrators to configure and manage their accounts. Usernames and passwords are securely stored, and the system includes password recovery options to enhance usability and security.

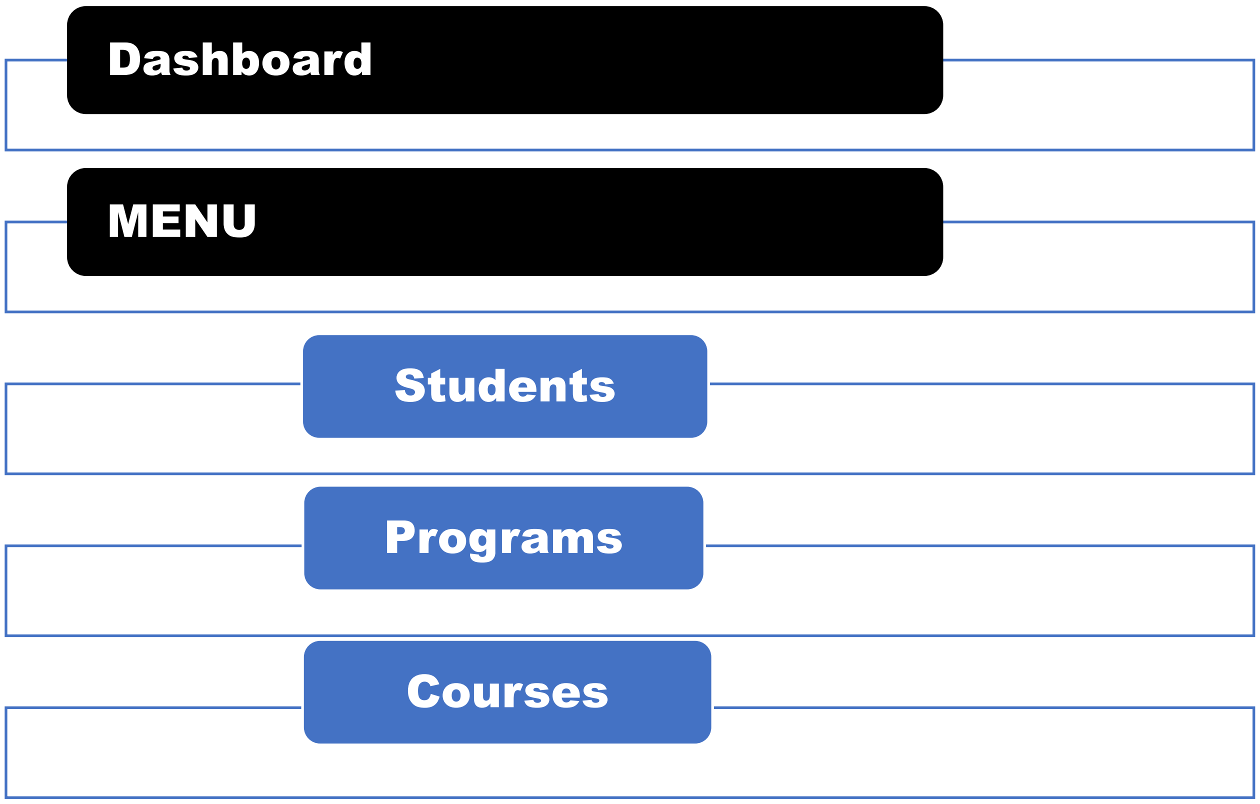
3.3.1 System Flowcharts and Diagrams:

Flowcharts and diagrams help to visualize the flow of data and interactions within each feature, offering a clear understanding of their processes and functionality.

1. **Menu Navigation Dataflow Diagram:**

This flowchart outlines the process of menu navigation. When an administrator launches the GIS, they are presented with a menu displaying options for Students, Programs, and Courses. By selecting one of these options, the user is directed to the corresponding module, where they can interact with the associated data.

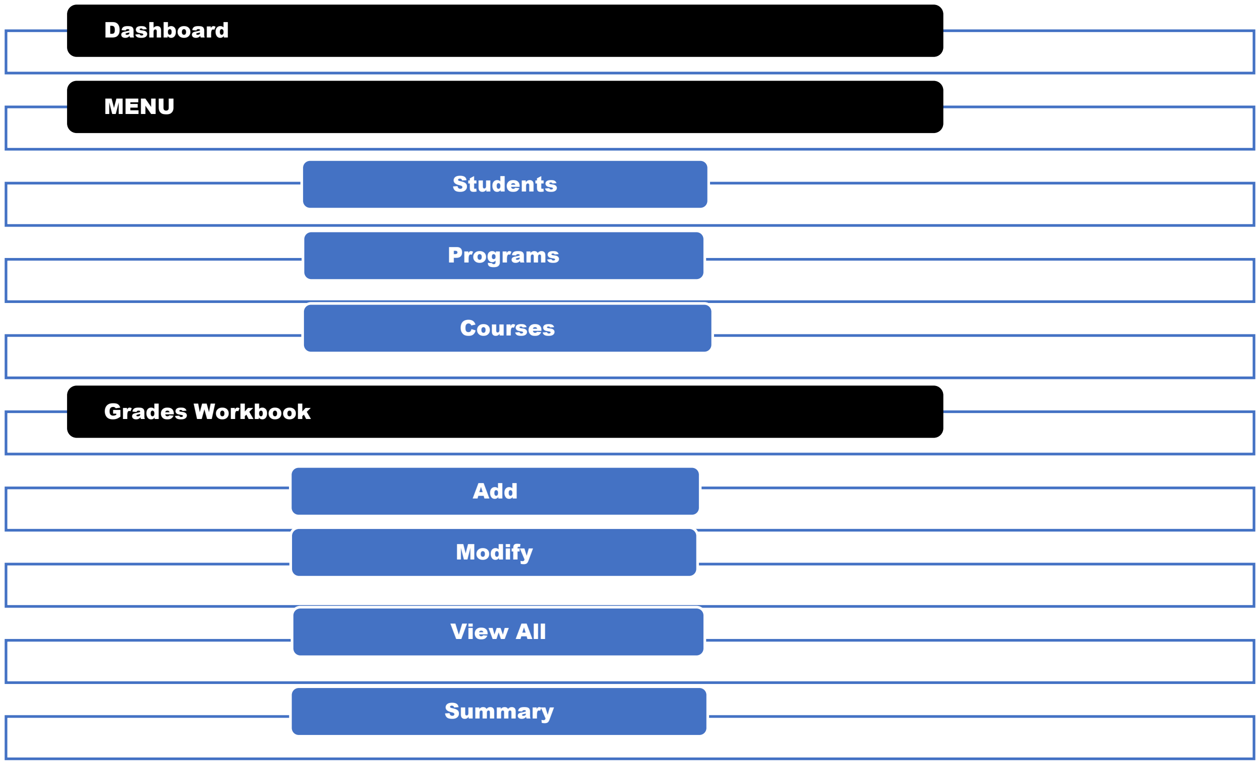
Figure 1. Menu Dataflow diagram



2. **Dashboard Dataflow Diagram:**

This diagram illustrates the data flow within the Dashboard feature. Data from the database, including student, program, and course information, is retrieved and presented on the dashboard in real time. The diagram shows how the database interacts with the user interface to provide up-to-date statistics.

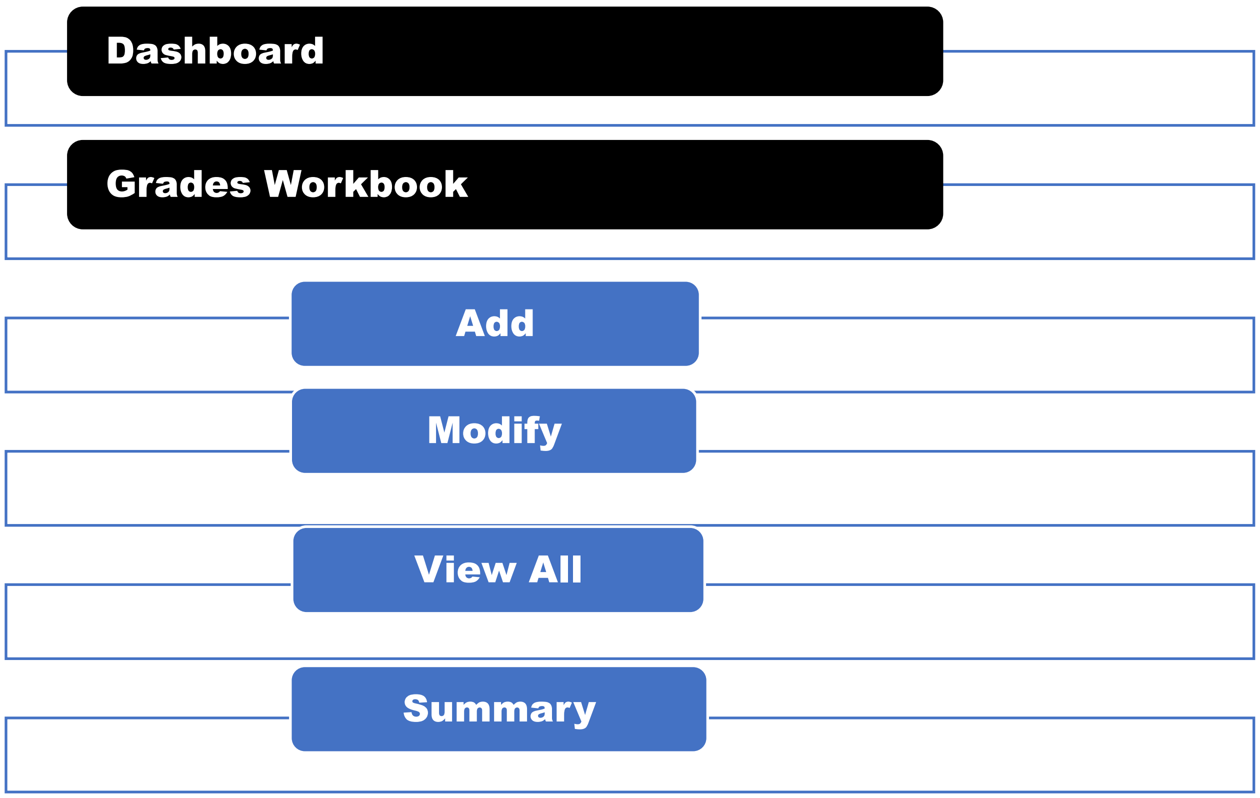
Figure 2. Dashboard Dataflow Diagram



3. Grades Workbook Flowchart: Adding Grading Data:

This flowchart details the process of adding grading data within the Grades Workbook feature. Administrators select the "Add" function, enter student and course details, input grades, and submit the data. The system performs validation checks to ensure accurate data entry.

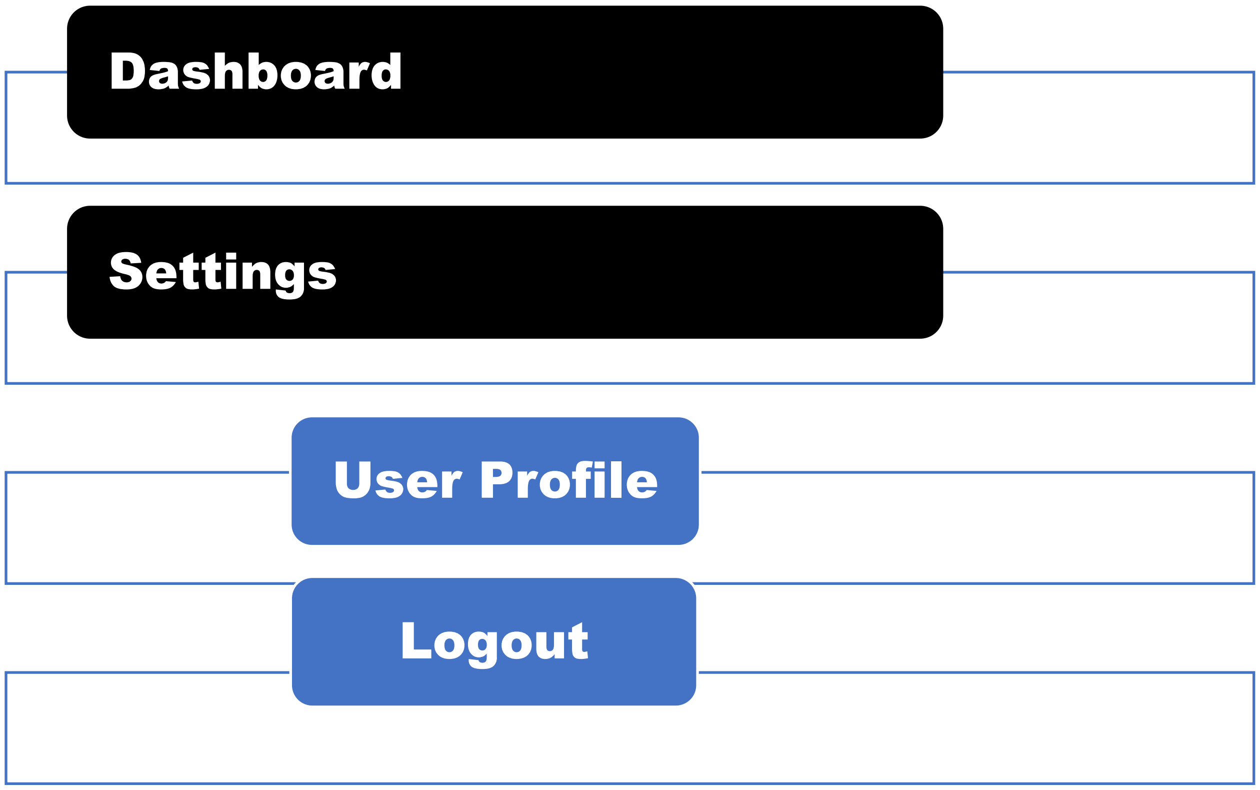
Figure 3. Grade Book Dataflow Diagram



4. **Settings Dataflow Diagram: User Account Management:**

This flowchart illustrates the process of managing user accounts within the Settings feature. Administrators can create, modify, or delete user accounts. The system validates user inputs and provides options for password recovery to enhance security and usability.

Figure 4. Setting Dataflow Diagram



3.3.2 Database Structure:

The structure of the database is instrumental in the functionality of each feature. The database schema is carefully designed to accommodate the unique requirements of Students, Programs, Courses, Grades, and User Accounts.

1. **Students Table:**

The Students Table stores student data, including student ID, name, date of birth, contact information, program enrollment, and other relevant details. The student ID serves as the primary key, ensuring the uniqueness of each student's identifier. Relationships are established with other tables, allowing for the association of students with programs and courses.

Table 1. Students Data Structure Table

Field Name	Datatype	Null	Default	Key	Extra
StudID	INTEGER	NOT NULL	NONE	PRIMARY KEY	AUTOINCREMENT
FirstName	TEXT	NOT NULL	NONE		
MiddleName	TEXT	NOT NULL	NONE		
LastName	TEXT	NOT NULL	NONE		
StudentID	INTEGER	NOT NULL	NONE		
ProgramID	INTEGER	NOT NULL	NONE	FOREIGN KEY	REFERENCES PROGRAM TABLE

2. Programs Table:

The Programs Table stores program information, including program ID, program name, and program coordinator. Program ID serves as the primary key, ensuring program identification. The table forms relationships with the Students Table and Courses Table to associate programs with students and available courses.

Table 2. Program Data Structure Table

Field Name	Datatype	Null	Default	Key	Extra
ProgramID	INTEGER	NOT NULL	NONE	PRIMARY KEY	AUTOINCREMENT
ProgramName	TEXT	NOT NULL	NONE		
ProgramLevel	TEXT	NOT NULL	NONE		

3. Courses Table:

The Courses Table contains course data, including course ID, course name, course instructor, and program affiliation. Course ID serves as the primary key, ensuring unique course identification. The table is related to the Programs Table to establish connections between courses and programs.

Table 3. Course Data Structure Table

Field Name	Datatype	Null	Default	Key	Extra
CourseID	INTEGER	NOT NULL	NONE	PRIMARY KEY	AUTOINCREMENT
CourseName	TEXT	NOT NULL	NONE		
CourseCode	TEXT	NOT NULL	NONE		
CreditHour	INTEGER	NOT NULL	NONE		
ProgramID	INTEGER	NOT NULL	NONE	FOREIGN KEY	REFERENCES TABLE PROGRAM
Year	INTEGER	NOT NULL	NONE		
Semester	TEXT	NOT NULL	NONE		

4. Grades Table:

The Grades Table is responsible for storing grading data, encompassing course ID, student ID, assignment scores, exam scores, and the final grade. Relationships are established with the Students Table and Courses Table to associate grading data with individual students and courses.

(a) Semester Grades table

Table 4. Semester Data Structure Table

Field Name	Datatype	Null	Default	Key	Extra
GradeID	INTEGER	NOT NULL	NONE	PRIMARY KEY	AUTOINCREMENT
StudID	INTEGER	NOT NULL	NONE	FOREIGN KEY	REFERENCES TABLE STUDENT DATA
CourseID	INTEGER	NOT NULL	NONE	FOREIGN KEY	REFERENCES TABLE COURSE DATA
Grade	TEXT	NOT NULL	NONE		
Semester	TEXT	NOT NULL	NONE		

(b) Cumulative Grades Table

Table 5. Cummulative Grades Data Structure Table

Field Name	Datatype	Null	Default	Key	Extra
AllGradesID	INTEGER	NOT NULL	NONE	PRIMARY KEY	AUTOINCREMENT
Grade1	INTEGER		NONE	FOREIGN KEY	REFERENCES TABLE STUDENT GRADE DATA
Grade2	INTEGER		NONE	FOREIGN KEY	REFERENCES TABLE STUDENT GRADE DATA
Grade3	INTEGER		NONE	FOREIGN KEY	REFERENCES TABLE STUDENT GRADE DATA
Grade4	INTEGER		NONE	FOREIGN KEY	REFERENCES TABLE STUDENT GRADE DATA
Grade5	INTEGER		NONE	FOREIGN KEY	REFERENCES TABLE STUDENT GRADE DATA
Grade6	INTEGER		NONE	FOREIGN KEY	REFERENCES TABLE STUDENT GRADE DATA
Grade7	INTEGER		NONE	FOREIGN KEY	REFERENCES TABLE STUDENT GRADE DATA
Grade8	INTEGER		NONE	FOREIGN KEY	REFERENCES TABLE STUDENT GRADE DATA
Grade9	INTEGER		NONE	FOREIGN KEY	REFERENCES TABLE STUDENT GRADE DATA
Grade10	INTEGER		NONE	FOREIGN KEY	REFERENCES TABLE STUDENT GRADE DATA
Grade11	INTEGER		NONE	FOREIGN KEY	REFERENCES TABLE STUDENT GRADE DATA
Grade12	INTEGER		NONE	FOREIGN KEY	REFERENCES TABLE STUDENT GRADE DATA
Grade13	INTEGER		NONE	FOREIGN KEY	REFERENCES TABLE STUDENT GRADE DATA
Grade14	INTEGER		NONE	FOREIGN KEY	REFERENCES TABLE STUDENT GRADE DATA
TotalCreditHours	INTEGER		NONE		
SGPA	REAL		NONE		
CGPA	REAL		NONE		
GradeComment	TEXT		NONE		

5. User Accounts Table:

The User Accounts Table is essential for managing user accounts. It includes fields for usernames, hashed passwords, and access privileges. Usernames serve as unique identifiers. This table is responsible for user authentication and access control.

Table 6. User Account Data Structure Table

Field Name	Datatype	Null	Default	Key	Extra
UserID	INTEGER	NOT NULL	NONE	PRIMARY KEY	AUTOINCREMENT
FirstName	TEXT	NOT NULL	NONE	FOREIGN KEY	
LastName	TEXT	NOT NULL	NONE	FOREIGN KEY	
SecretPin	TEXT	NOT NULL	NONE	FOREIGN KEY	
UserName	TEXT	NOT NULL	NONE	FOREIGN KEY	
Password	TEXT	NOT NULL	NONE	FOREIGN KEY	
UserRole	TEXT	NOT NULL	NONE	FOREIGN KEY	
ImagePath	INTEGER	NOT NULL	NONE	FOREIGN KEY	

The relationships between tables ensure data integrity and allow for efficient data retrieval and management. The database structure is a cornerstone of the GIS, enabling the effective storage and retrieval of data associated with each feature

3.4 System Architecture

System Components:

The architecture of the Grading Information System consists of multiple components, each playing a crucial role in delivering the functionality of the Menu, Dashboard, Grades Workbook, and Settings.

1. **User Interface Layer:**

The user interface layer encompasses the front-end of the system, providing the visual and interactive components that users interact with. It includes forms, menus, buttons, and controls for data entry, retrieval, and navigation. The user interface is tailored to the specific needs of each feature, ensuring a user-friendly experience.

2. **Logic Layer:**

The logic layer serves as the core functionality of the GIS, responsible for data processing, validation, and database interactions. It ensures that data entered by administrators is validated, processed, and stored accurately. This layer also includes the logic for calculations, such as determining final grades.

3. **Data Access Layer:**

The data access layer manages interactions with the SQLite database. It includes functions for data retrieval, storage, and manipulation. This layer is pivotal for maintaining data consistency and integrity. When administrators interact with the system, the data access layer handles data requests and updates to the database.

4. **Security Layer:**

The security layer is responsible for safeguarding user accounts and data. It includes mechanisms for user authentication and authorization, ensuring that only authorized individuals gain access to the system. User account data, including usernames and passwords, is securely managed within this layer, providing an additional layer of protection.

3.4.1 Data Flow and Processing:

Each feature within the Grading Information System follows a structured flow of data and processing to meet the needs of department administrators.

1. **The Menu Feature:**

When a department administrator logs in, the system verifies their credentials through the security layer. Upon successful authentication, the user gains access to the Menu feature. The Menu interface is presented, allowing the administrator to select from the available modules: Students, Programs, and Courses. When a module is selected, the system queries the database to retrieve the relevant data for display.

2. **The Dashboard Feature:**

The Dashboard feature relies on the data access layer to retrieve real-time statistics from the database. When a department administrator accesses the Dashboard, the system initiates database queries to fetch data related to the total number of students, programs, and courses. This data is then displayed in the user interface layer in a visually engaging format.

3. **The Grades Workbook Feature:**

Administrators use the Grades Workbook feature to add, modify, view all, and summarize grading data. When administrators input data, the business logic layer handles the validation and processing. For instance, when adding grades, the system validates that the data entered adheres to specified criteria. Data is then stored in the database through the data access layer. When administrators request summaries of grading data, the business logic layer performs the necessary calculations, such as determining final grades.

4. **The Settings Feature:**

The Settings feature encompasses user account management, including the creation, modification, and deletion of user accounts. When administrators interact with the Settings feature, the security layer manages user authentication and authorization. User account details are stored securely in the database. The system provides options for password recovery to enhance usability and security.

3.4.2 Security and Data Integrity:

Security measures are paramount in the Grading Information System to protect sensitive student data and maintain data integrity.

1. **User Authentication:**

The system employs a robust authentication mechanism to verify the identity of department administrators. Passwords are securely hashed and stored in the database, rendering them unreadable in the event of unauthorized access. When administrators log in, the security layer ensures that only valid credentials gain access to the system. This authentication mechanism protects sensitive data and maintains user account security.

2. **Data Validation:**

Data validation is a crucial aspect of the system's design. When administrators interact with the GIS, the business logic layer checks user inputs for valid formats, ranges, and dependencies. For example, when adding student data or grading information, the system ensures that fields are filled correctly and that data adheres to specified criteria. Validation checks help prevent the storage of erroneous or inconsistent data, enhancing data accuracy.

3. **Encryption:**

Sensitive data, including user passwords, is encrypted to protect it from unauthorized access. The system uses encryption algorithms to transform passwords into a secure format that is indecipherable without the appropriate decryption key. This encryption method adds an additional layer of security to user account data.

4. **Error Handling:**

The Grading Information System incorporates comprehensive error handling mechanisms to gracefully manage exceptions and prevent data corruption. When errors occur, informative error messages are generated and displayed to administrators. These messages guide users in resolving issues and ensure that the system continues to function smoothly.

3.5 Summary

In summary, Chapter three has provided a comprehensive exploration of the Grading Information System's design and architecture, focusing on the integration of specific features: Menu, Dashboard, Grades Workbook, and Settings.

The design of the user interface for each feature prioritizes user-friendliness and efficient data entry and retrieval. Flowcharts and diagrams have been used to visually represent the flow of data and interactions within the system. The database structure, featuring tables for students, programs, courses, grades, and user accounts, ensures the efficient storage and retrieval of data.

The system architecture comprises multiple layers, including the user interface layer, business logic layer, data access layer, and security layer, each contributing to the functionality and security of the GIS. Data flows seamlessly within the system, and a range of security measures, including user authentication, data validation, encryption, error handling, and an audit trail, contribute to the system's robustness.

The Grading Information System's design and architecture empower the Menu, Dashboard, Grades Workbook, and Settings features to meet the specific needs of department administrators efficiently. These features have been strategically integrated into the application, providing a powerful tool for streamlining the grading process and ensuring data security.

CHAPTER FOUR: SYSTEM IMPLEMENTATION

Introduction

The successful implementation of the Grading Information System (GIS) is a crucial milestone in the development process. This chapter delves into the practical aspects of how the system was brought to life, covering the development environment, coding of major features, database integration, user account management, testing, documentation, and the deployment of the system.

4.0 Development Environment

The implementation of the GIS was made possible through a carefully chosen development environment that supported the use of Visual Basic as the primary programming language. Visual Basic provided a robust platform for creating the system, offering a user-friendly integrated development environment (IDE) for designing and coding the user interface. Version control tools were utilized to manage the project, ensuring that code changes and project files were tracked and controlled.

4.1 System Components Implementation

Each of the Grading Information System's core components, including the Menu, Dashboard, Grades Workbook, and Settings features, was meticulously implemented.

1. **Dashboard Feature:**

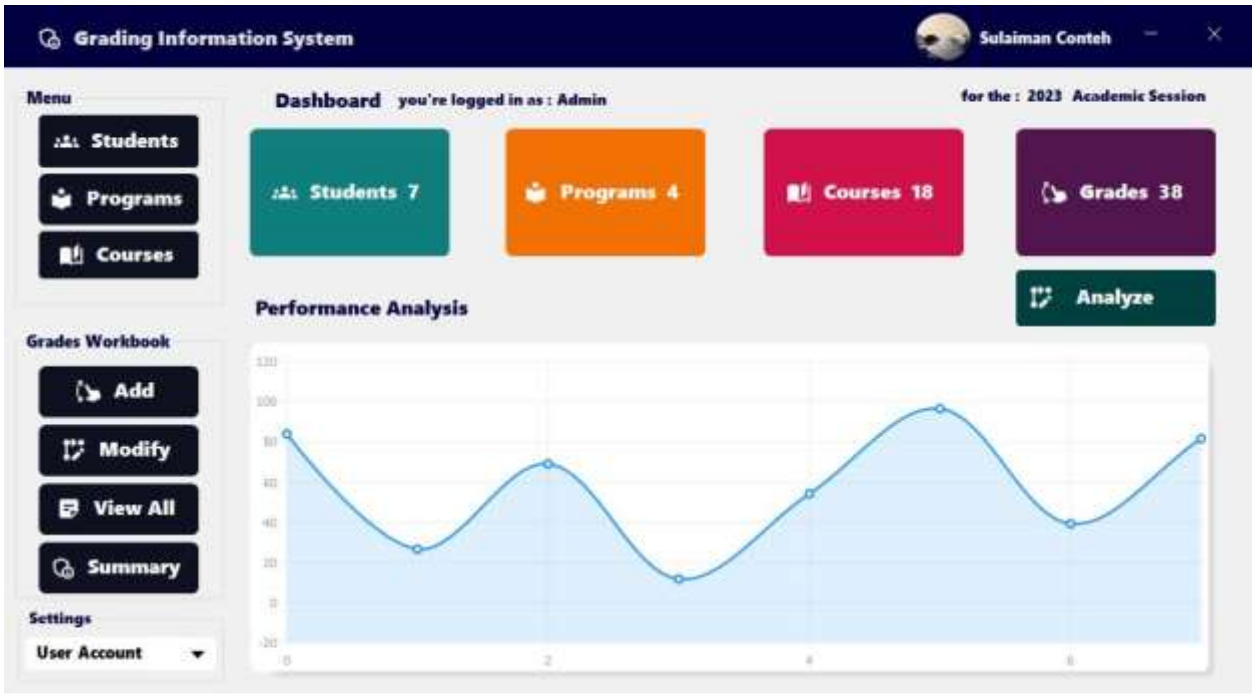


Figure 5. Dashboard Feature

The Dashboard feature offers a real-time overview of the system's data, presenting the total number of students, programs, and courses. This screenshot showcases the visual appeal and functionality of the Dashboard, providing administrators with insights into the department's composition.

2. **Menu Feature:**

The Menu feature serves as the system's main navigation hub. Administrators can effortlessly access the Students, Programs, and Courses modules. This screenshot provides a visual representation of the Menu's design, demonstrating its user-friendly layout and accessibility.

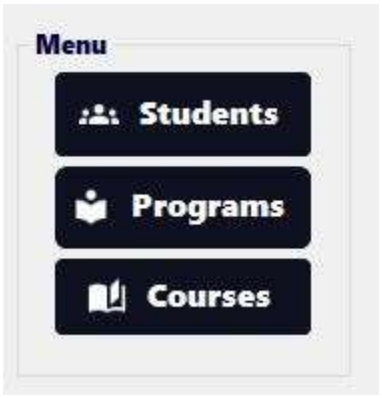
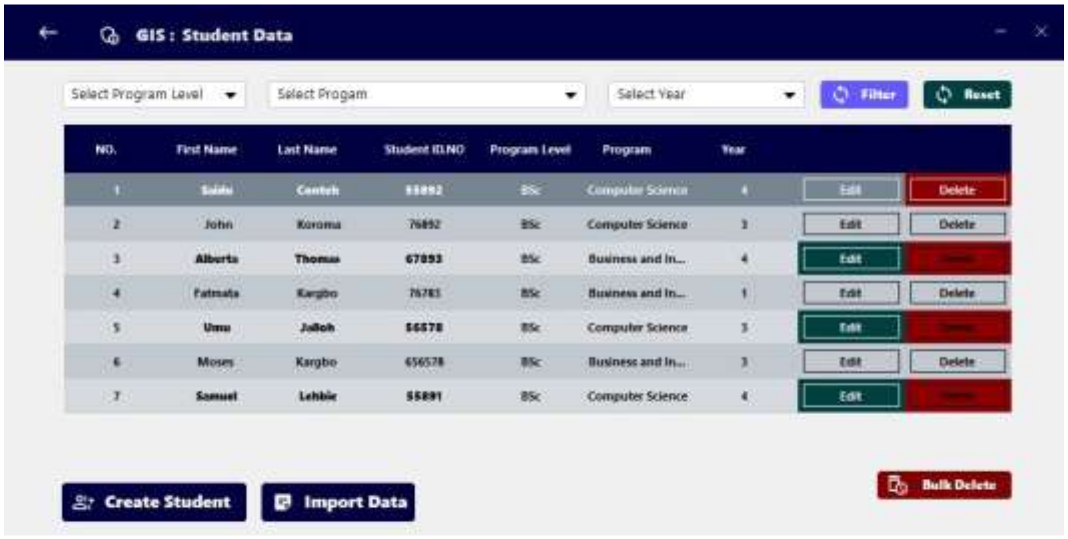


Figure 6. Menu Feature

(a) Students

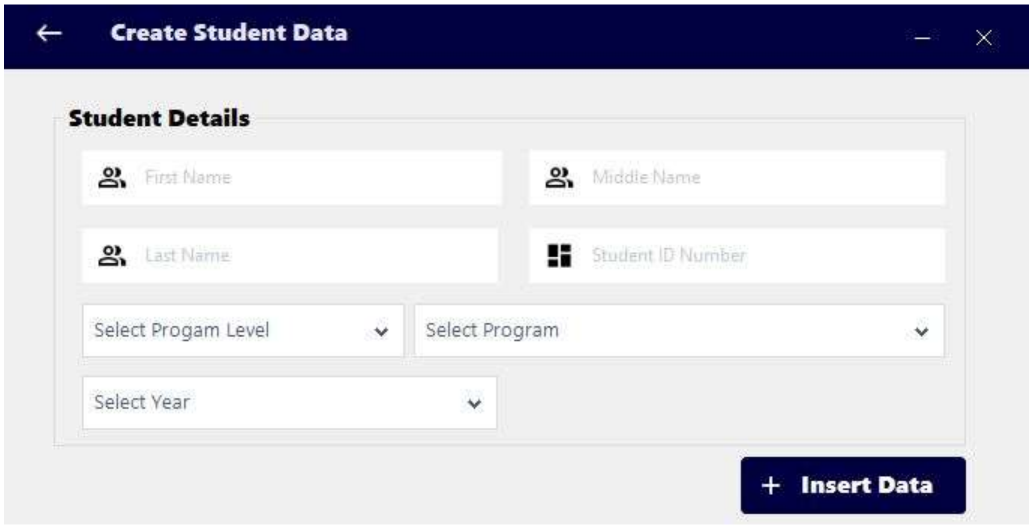
1. **General View**



NO.	First Name	Last Name	Student ID.NO	Program Level	Program	Year		
1	Saidu	Conteh	55892	BSc	Computer Science	4	Edit	Delete
2	John	Koroma	76892	BSc	Computer Science	3	Edit	Delete
3	Alberta	Thomas	67893	BSc	Business and In...	4	Edit	[illegible]
4	Fatmata	Kargbo	76783	BSc	Business and In...	1	Edit	Delete
5	Umu	Jalloh	56578	BSc	Computer Science	3	Edit	[illegible]
6	Moses	Kargbo	656578	BSc	Business and In...	3	Edit	Delete
7	Samuel	Lehbie	55891	BSc	Computer Science	4	Edit	[illegible]

Figure 7. General Student View

2. **Create new student**


Create Student Data

Student Details

First Name

Middle Name

Last Name

Student ID Number

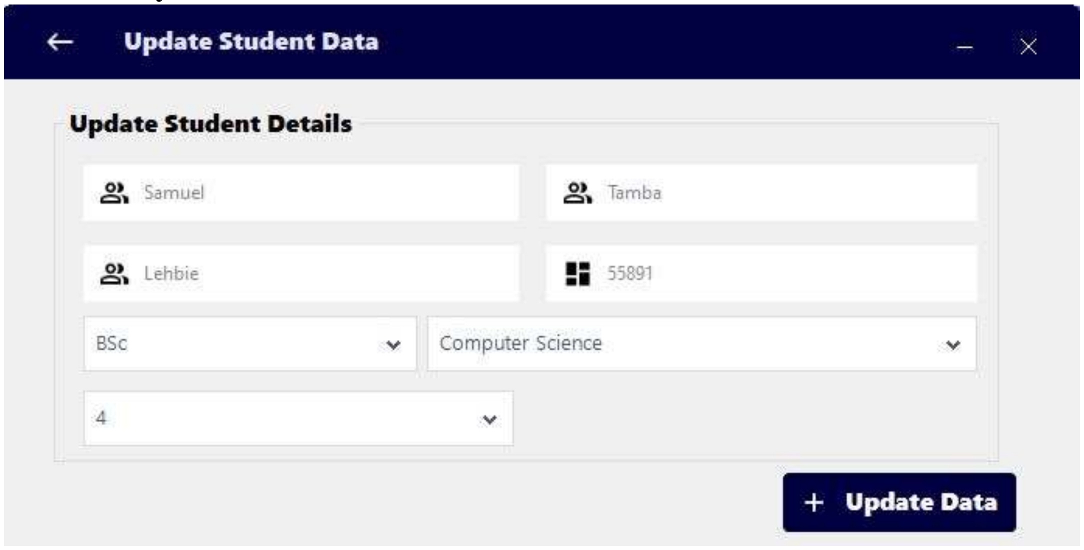
Select Progam Level

Select Program

Select Year

Figure 8. Create New Student View

3. **Edit/Modify Student**


Update Student Data

Update Student Details

Samuel

Tamba

Lehbie

55891

BSc

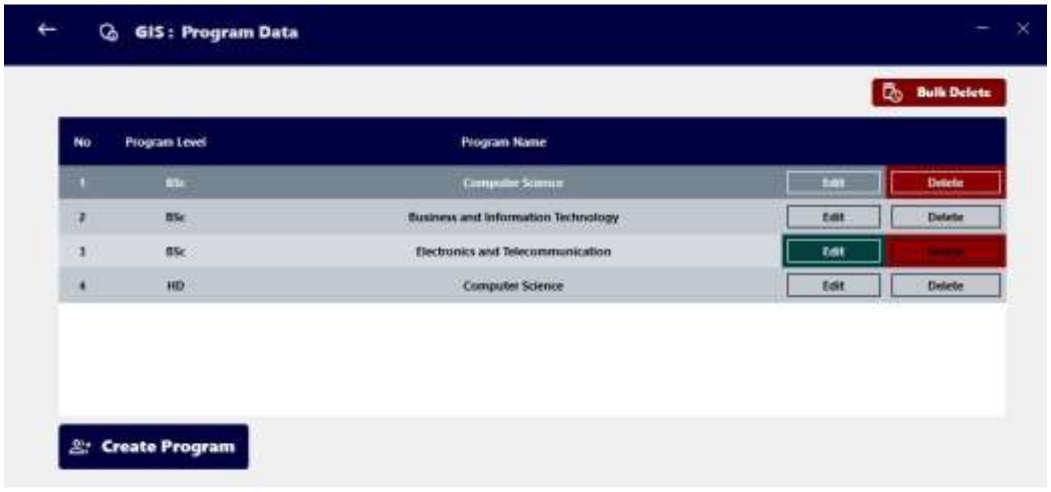
Computer Science

4

Figure 9. Edit/Modify Student View

(b) Programs

1. **General View**



No	Program Level	Program Name		
1	BSc	Computer Science	Edit	Delete
2	BSc	Business and Information Technology	Edit	Delete
3	BSc	Electronics and Telecommunication	Edit	Delete
4	HD	Computer Science	Edit	Delete

Figure 10. Program General View

2. **Create Program**

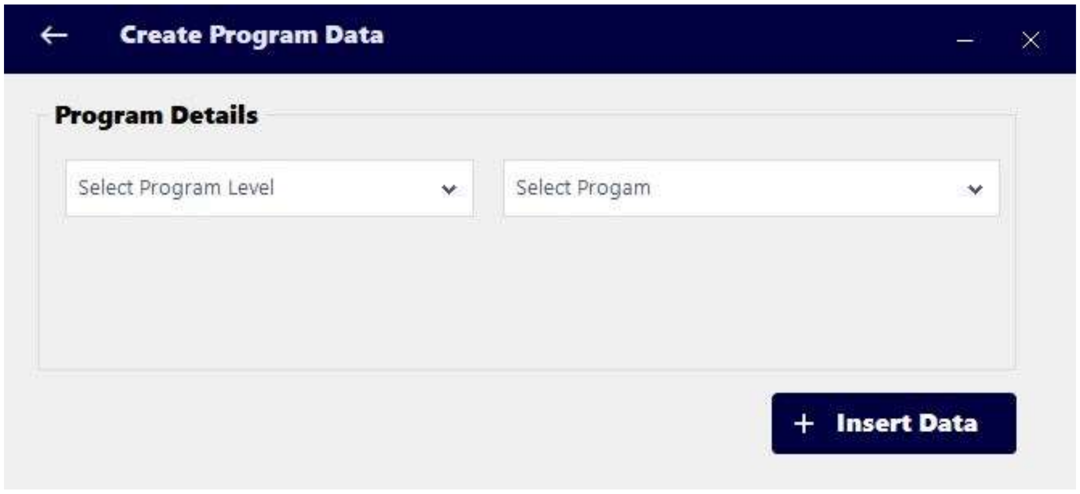


Figure 11. Create Program View

3. **Edit/Modify Program**

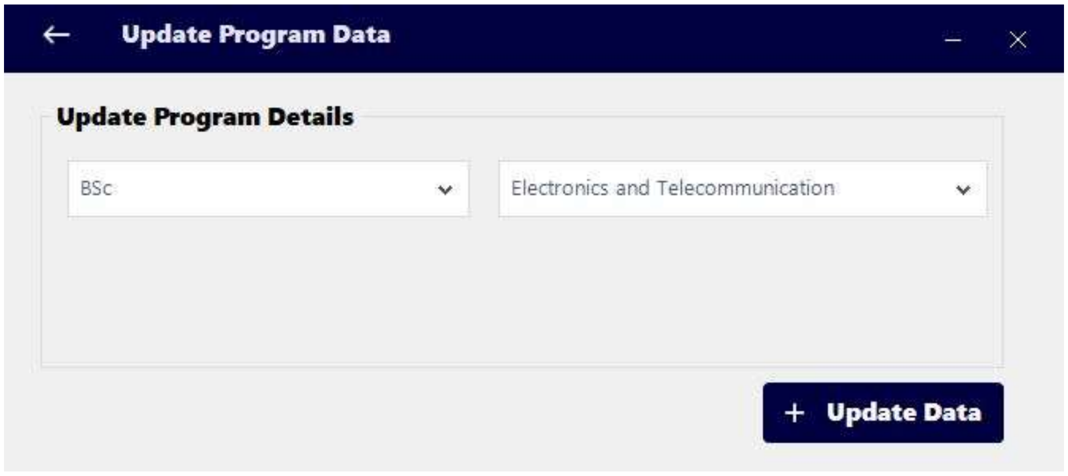
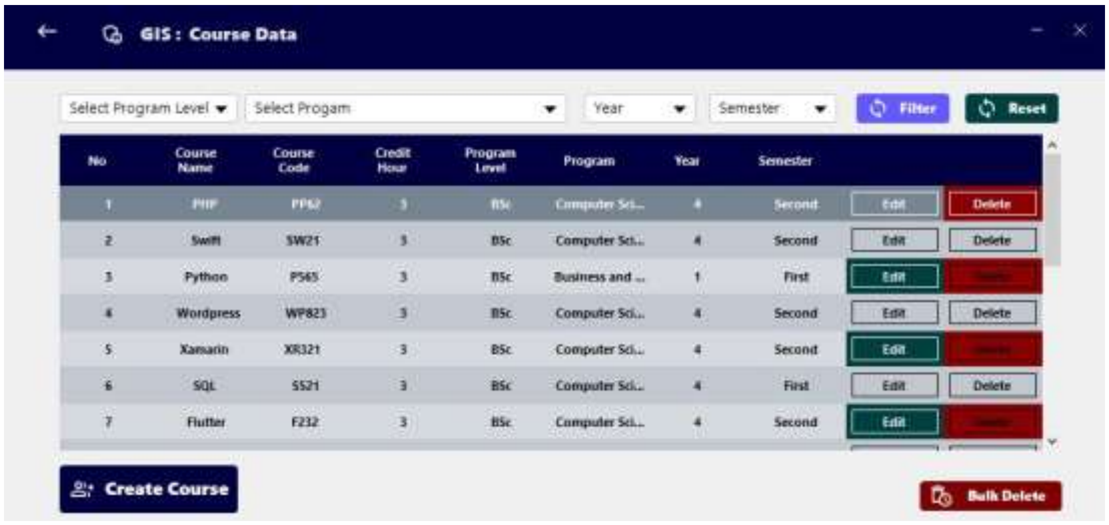


Figure 12. Edit/Modify Program View

(c) Courses

1. **General View**



No	Course Name	Course Code	Credit Hour	Program Level	Program	Year	Semester		
1	PHP	PP62	3	BSc	Computer Sci...	4	Second	Edit	Delete
2	Swift	SW21	3	BSc	Computer Sci...	4	Second	Edit	Delete
3	Python	P565	3	BSc	Business and ...	1	First	Edit	[illegible]
4	Wordpress	WP823	3	BSc	Computer Sci...	4	Second	Edit	Delete
5	Xamarin	XR321	3	BSc	Computer Sci...	4	Second	Edit	[illegible]
6	SQL	S521	3	BSc	Computer Sci...	4	First	Edit	Delete
7	Flutter	F232	3	BSc	Computer Sci...	4	Second	Edit	[illegible]

Figure 13. Courses General View

2. **Create new course**

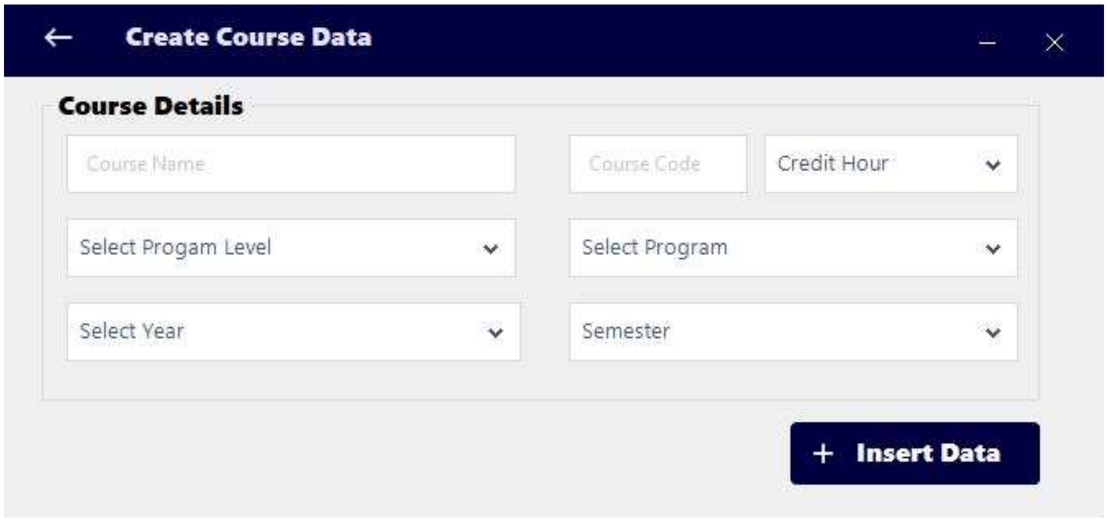
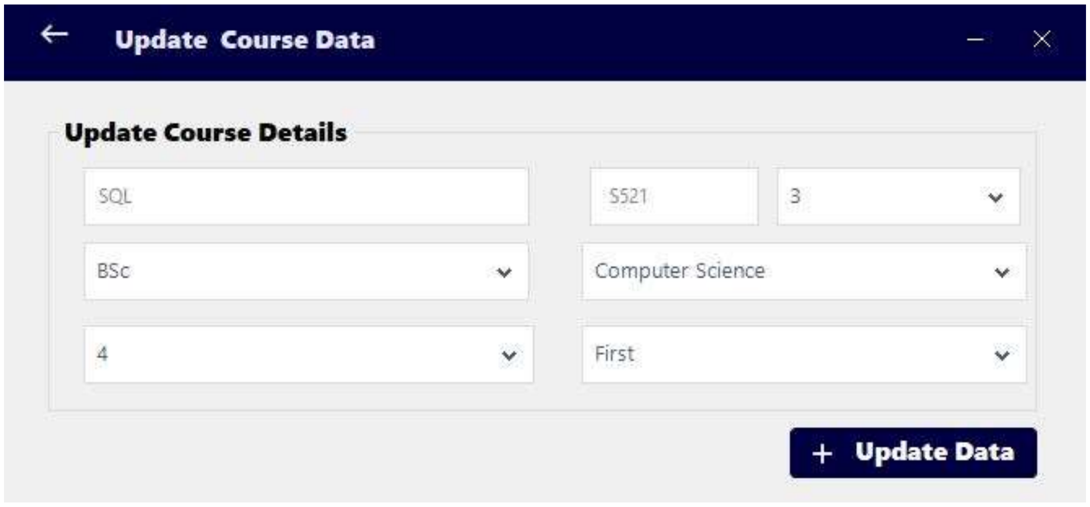


Figure 14. Create New Course View

3. **Edit/Modify Course**



SQL	S521	3
BSc	Computer Science	
4	First	

Figure 15. Edit/Modify Course View

3. **Grades Workbook Feature:**

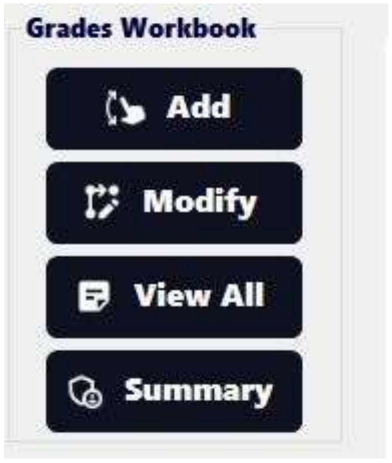
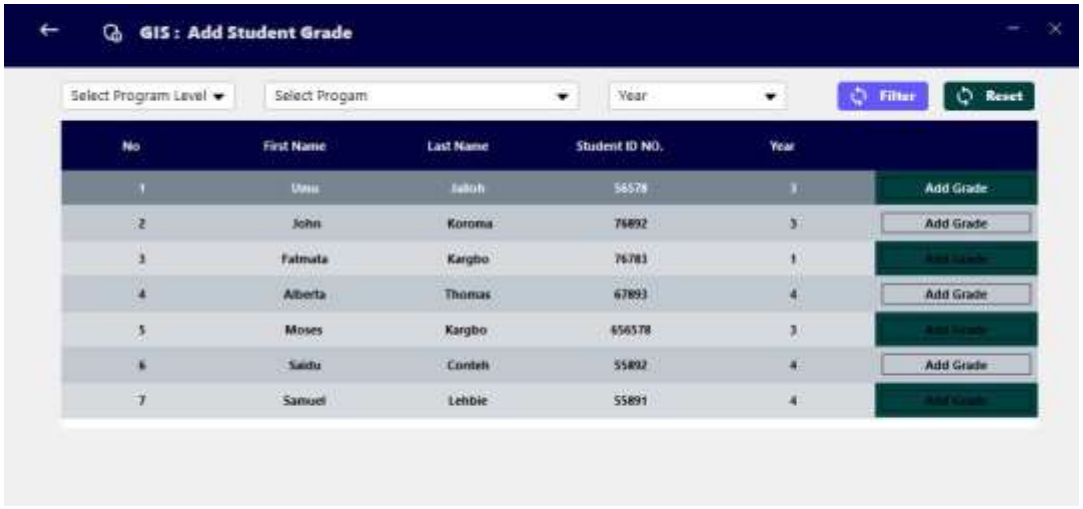


Figure 16. Grade Book *View*

The Grades Workbook feature enables administrators to add, modify, view all, and summarize grading data. This screenshot illustrates how the feature streamlines the grading process, offering user-friendly input forms and feedback mechanisms.

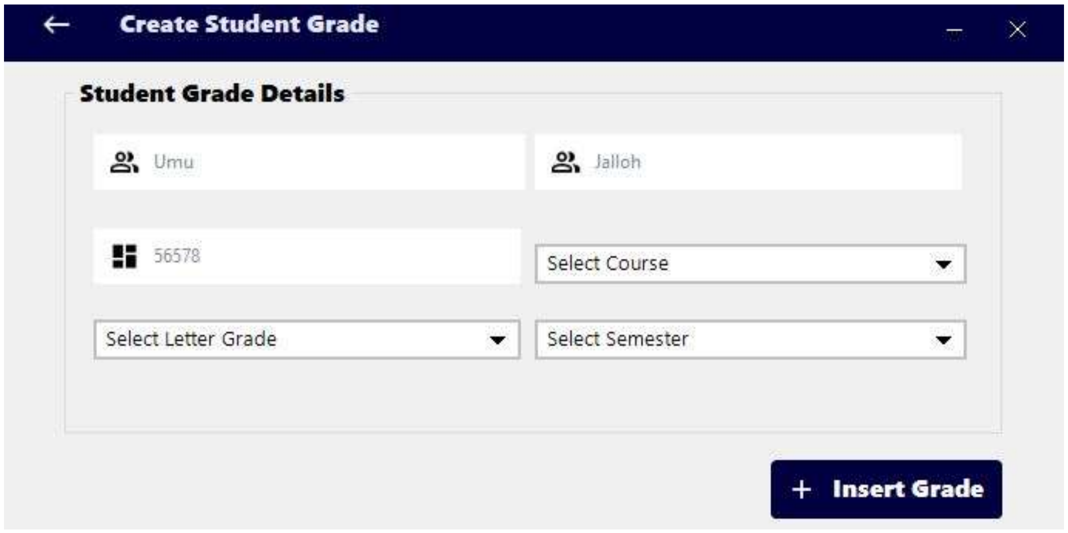
(a) General View



No	First Name	Last Name	Student ID NO.	Year	
1	Umu	Jalloh	56578	3	Add Grade
2	John	Koroma	76892	3	Add Grade
3	Fatmata	Kargbo	76783	1	Add Grade
4	Alberta	Thomas	67893	4	Add Grade
5	Moses	Kargbo	656578	3	Add Grade
6	Saidu	Conteh	55892	4	Add Grade
7	Samuel	Lehbie	55891	4	Add Grade

Figure 17. Grade Workbook General View

(b) Add Student Grade



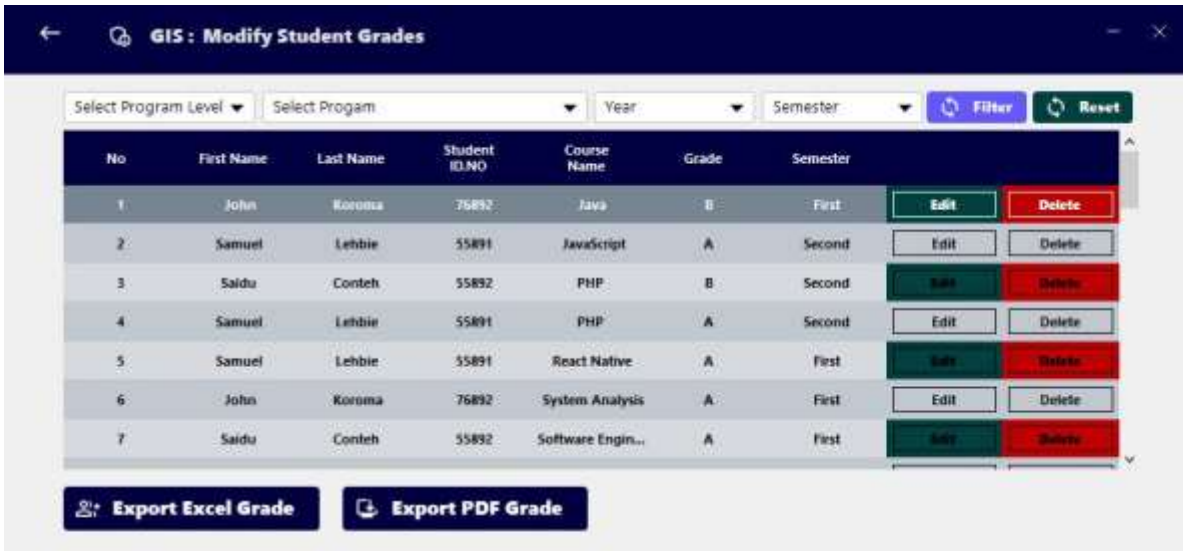
Umu

Jalloh

56578

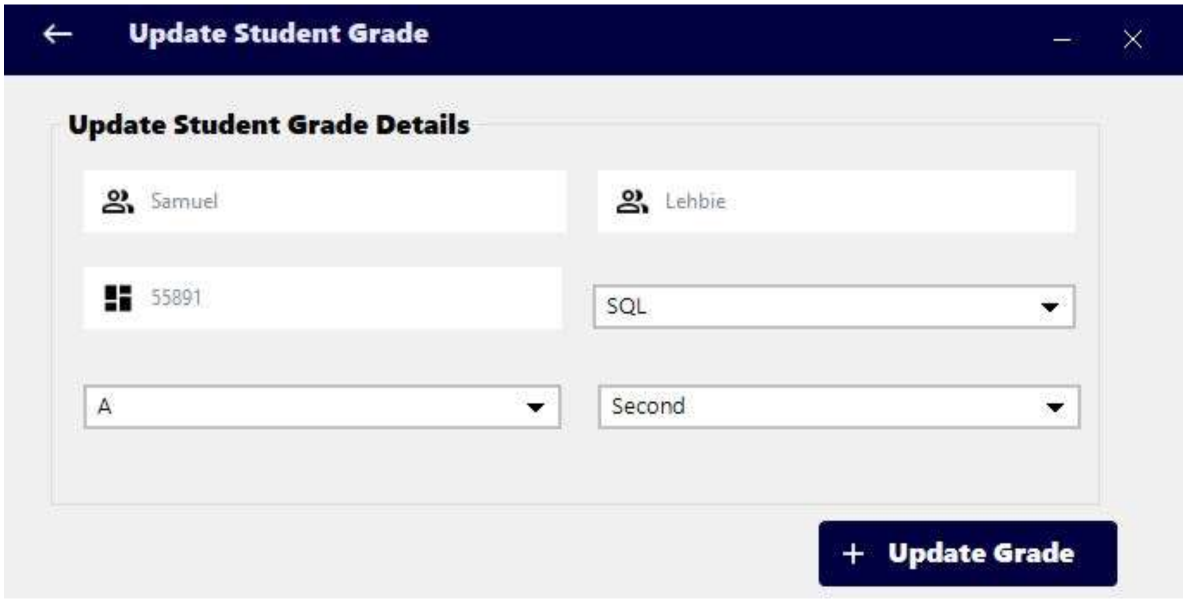
Figure 18. Add Student Grade View

(c) Modify/Edit Student Grades



No	First Name	Last Name	Student ID.NO	Course Name	Grade	Semester		
1	John	Koroma	76892	Java	B	First	Edit	Delete
2	Samuel	Lehbie	55891	JavaScript	A	Second	Edit	Delete
3	Saidu	Conteh	55892	PHP	B	Second	Edit	Delete
4	Samuel	Lehbie	55891	PHP	A	Second	Edit	Delete
5	Samuel	Lehbie	55891	React Native	A	First	Edit	Delete
6	John	Koroma	76892	System Analysis	A	First	Edit	Delete
7	Saidu	Conteh	55892	Software Engin...	A	First	Edit	Delete

Figure 19. Modify/Edit Student Grades View

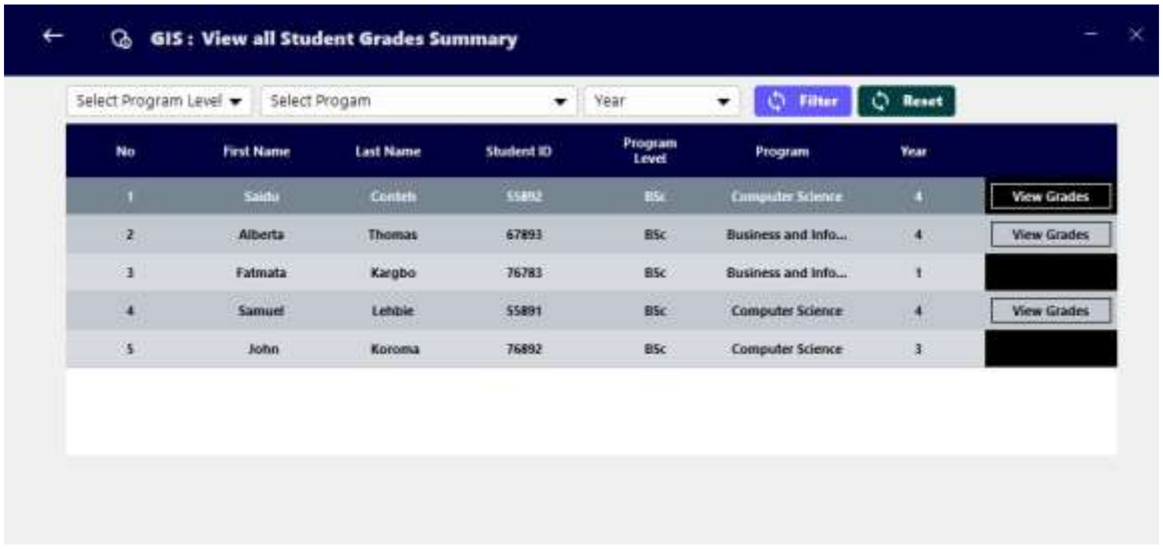


Samuel	Lehbie
55891	SQL
A	Second

Figure 20. Update Student Grade View

(d) View All Student Grades

1. **General View**



GIS : View all Student Grades Summary

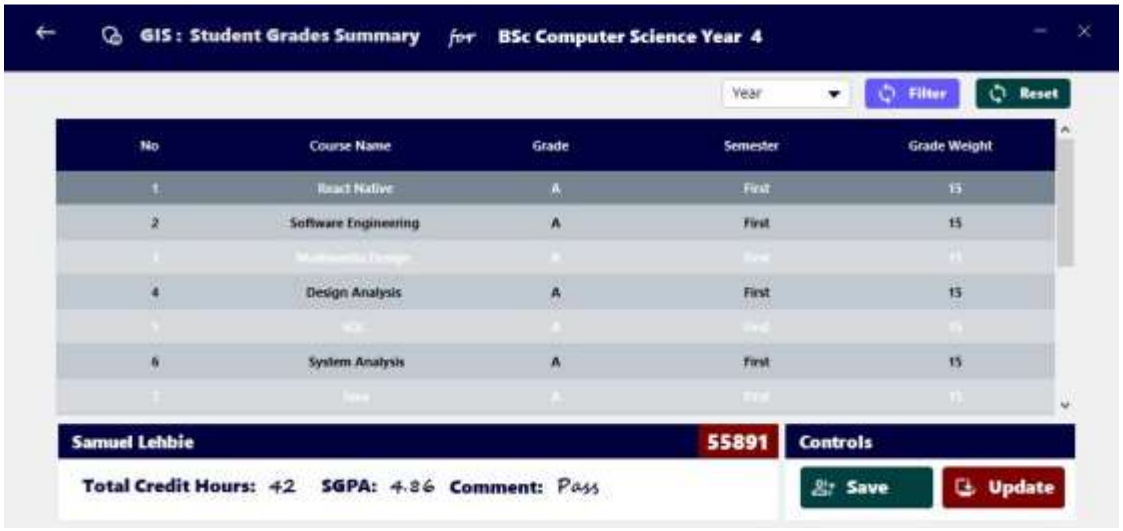
Select Program Level | Select Progam | Year | Filter | Reset

No	First Name	Last Name	Student ID	Program Level	Program	Year	
1	Saidu	Conteh	55892	BSc	Computer Science	4	View Grades
2	Alberta	Thomas	67893	BSc	Business and Info...	4	View Grades
3	Fatmata	Kargbo	76783	BSc	Business and Info...	1	
4	Samuel	Lehbie	55891	BSc	Computer Science	4	View Grades
5	John	Koroma	76892	BSc	Computer Science	3	

Figure 21. View All Student Grades General View

2. **Individual Student Grades View**

   (a) First Semester



GIS : Student Grades Summary *for* BSc Computer Science Year 4

Year | Filter | Reset

No	Course Name	Grade	Semester	Grade Weight
1	React Native	A	First	15
2	Software Engineering	A	First	15
3	Multimedia Design	A	First	15
4	Design Analysis	A	First	15
5	SQL	A	First	15
6	System Analysis	A	First	15
7	Java	A	First	15

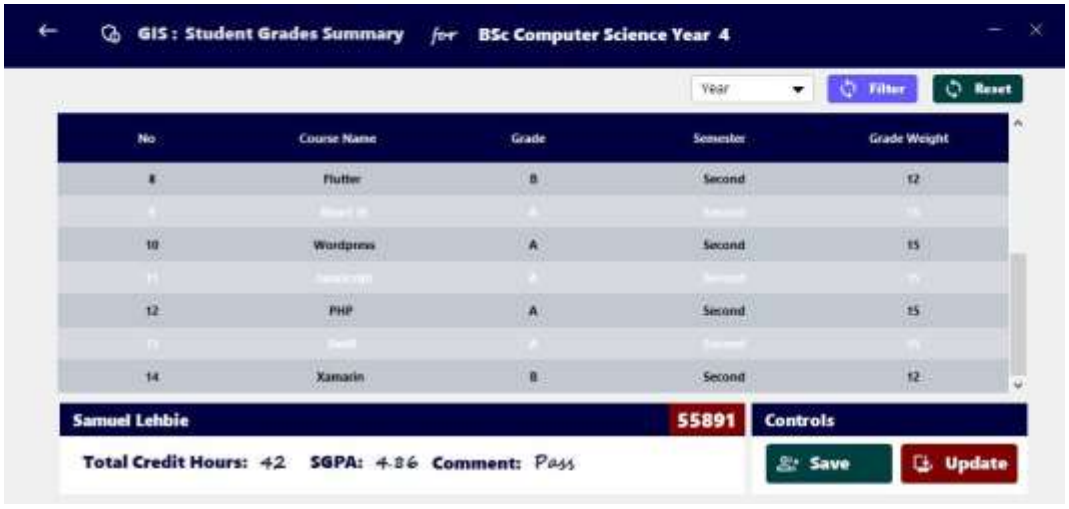
Samuel Lehbie 55891

Total Credit Hours: 42 SGPA: 4.86 Comment: Pass

Controls: Save | Update

Figure 22. Individual Student Grades View for First Semester

(b) Second Semester



GIS : Student Grades Summary *for* BSc Computer Science Year 4

No	Course Name	Grade	Semester	Grade Weight
8	Flutter	B	Second	12
9	[illegible]	A	Second	15
10	Wordpress	A	Second	15
11	Javascript	A	Second	15
12	PHP	A	Second	15
13	Swift	A	Second	15
14	Xamarin	B	Second	12

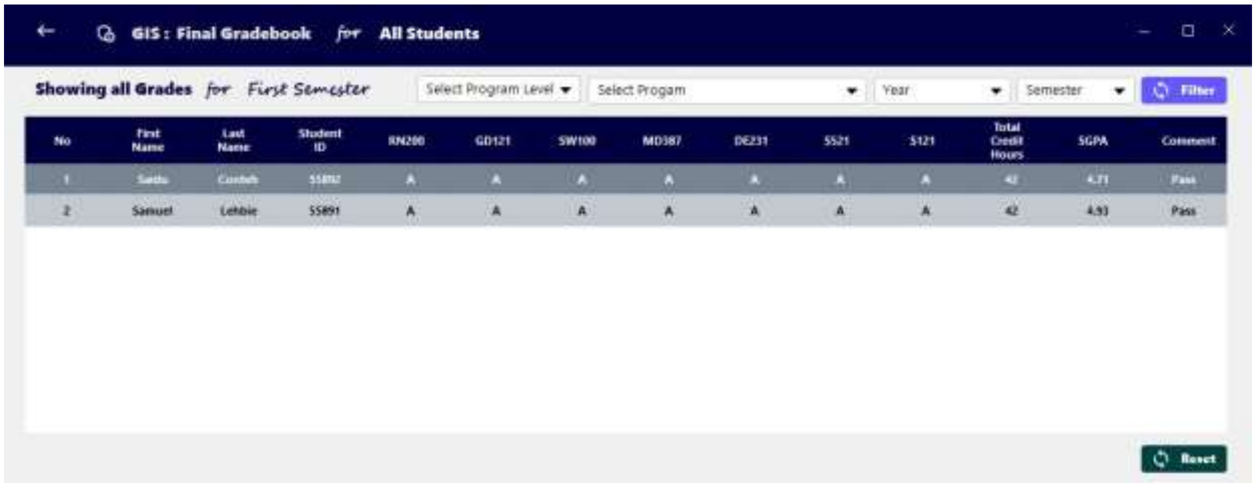
Samuel Lehbie 55891

Total Credit Hours: 42 SGPA: 4.86 Comment: Pass

Figure 23. Individual Student Grades View for Second Semester View

(e) Summary – provides an overall view of all student calculated grades

1. First Semester



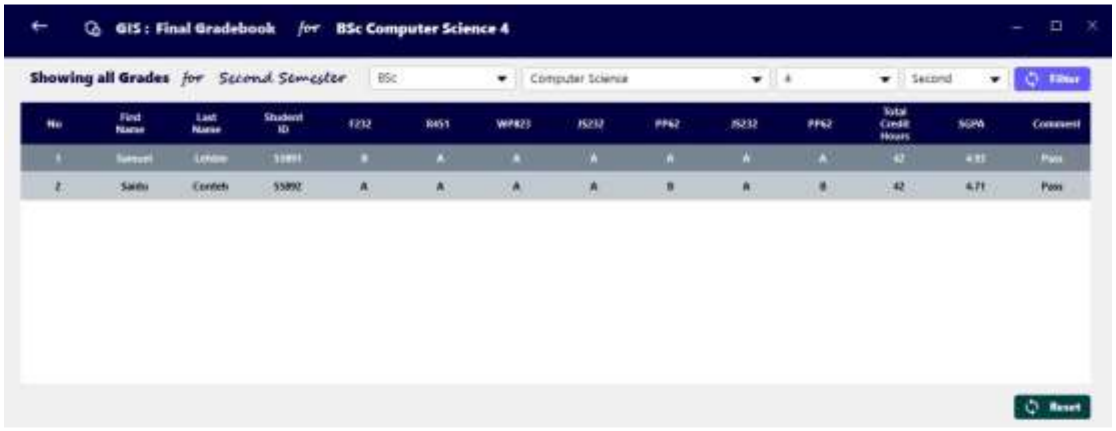
GIS : Final Gradebook *for* All Students

Showing all Grades *for* First Semester

No	First Name	Last Name	Student ID	RN200	GD121	SW100	MD387	DE231	S521	S121	Total Credit Hours	SGPA	Comment
1	Saidu	Conteh	55892	A	A	A	A	A	A	A	42	4.71	Pass
2	Samuel	Lehbie	55891	A	A	A	A	A	A	A	42	4.93	Pass

Figure 24. Summary View for First Semester View

2. **Second Semester**


GIS : Final Gradebook *for* BSc Computer Science 4

Showing all Grades *for* Second Semester BSc Computer Science 4 Second

No	First Name	Last Name	Student ID	F232	R451	WPR23	JS232	PP62	JS232	PP62	Total Credit Hours	SGPA	Comment
1	Samuel	Lebbie	55891	B	A	A	A	A	A	A	42	4.93	Pass
2	Saidu	Conteh	55892	A	A	A	A	B	A	B	42	4.71	Pass

Figure 25. Summary View for Second Semester View

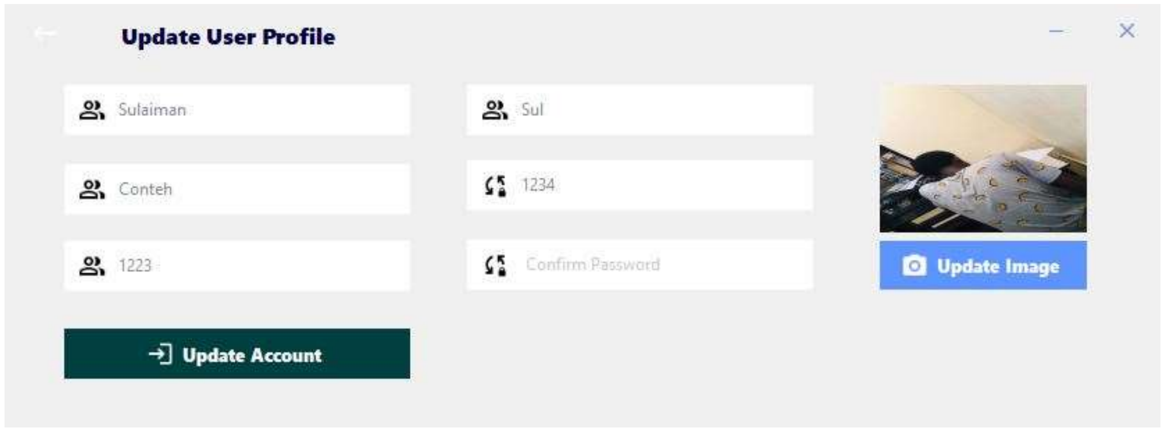
4. **Settings Feature:**

Figure 26. Setting View

The Settings feature is responsible for user account management. This screenshot exemplifies the interface used for creating, modifying, and deleting user accounts. It emphasizes the importance of secure user authentication and access control.

Each screenshot is accompanied by explanatory text that highlights the key features and functionalities of the respective interface.

(a) **User Account Profile**


Update User Profile

Sulaiman

Sul

Conteh

1234

1223

Confirm Password

Figure 27. User Account Profile View

4.2 Database Integration

The integration of the SQLite database into the system was a pivotal step in the implementation process. The database was created to store student, program, course, grading, and user account data. Tables and relationships were established to maintain data integrity. The data access layer facilitated efficient data retrieval, storage, and manipulation.

4.3 User Account Management

The Settings feature implemented user account management, covering user registration, password hashing, access control, and user authentication. This section details the implementation of user account management and its significance in ensuring system security.

4.4 Testing and Quality Assurance

Thorough testing was conducted to validate the functionality, security, and performance of the GIS. Various types of testing were employed, including unit testing, integration testing, and security testing. Test cases were designed and executed, leading to valuable findings and insights. Results, issues identified, and their resolutions are discussed.

CHAPTER FIVE: CONCLUSIONS AND RECCOMEDATION

Introduction

The Grading Information System project has been a journey of innovation and transformation. This chapter marks the conclusion of this endeavor, where we summarize the key findings, reflect on the system's impact, and offer recommendations for the future.

The Grading Information System was designed and implemented with a clear set of objectives in mind. It aimed to modernize and streamline the department's grading process, replacing the labor-intensive Excel-based system. Throughout the project's lifecycle, key features, including the Menu, Dashboard, Grades Workbook, and Settings, were meticulously developed to meet the department's unique requirements.

However, no project is without its challenges. During the implementation phase, we encountered issues related to real-time data retrieval and dynamic data presentation. These challenges led to the optimization of database queries and the improvement of data access mechanisms. Despite these hurdles, the development team's dedication and collaborative efforts resulted in a fully functional system.

5.0 Conclusions

The implementation of the Grading Information System has yielded significant positive outcomes for the department. By replacing the manual grading process with an automated desktop application, the department administrators have experienced increased efficiency, reduced errors, and improved data security. The Menu feature offers an intuitive gateway to student, program, and course data, while the Dashboard provides real-time insights into the department's composition. The Grades Workbook feature streamlines the grading process, making it faster and more accurate. The Settings feature ensures data security and access control, safeguarding sensitive information.

The impact of the Grading Information System goes beyond the technical aspects. It has revolutionized how the department manages grading, enhancing the overall administrative process. By automating many of the manual tasks involved in grading, it has reduced the workload and increased the accuracy of data. The system's real-time data presentation ensures that department administrators always have access to the most current information, providing a solid foundation for decision-making.

The user account management system in the Settings feature has also improved data security. User authentication and authorization mechanisms have ensured that only authorized personnel can access the system. The convenience of password recovery options further enhances the user experience.

5.1 Recommendations

While the Grading Information System has been a significant step forward, there is always room for improvement and expansion. As such, we offer several recommendations for the future:

1. Regular Updates: Continuously update and maintain the system to keep it in line with evolving department requirements and technological advancements.
2. Enhanced User Training: Develop comprehensive training programs for department administrators to ensure they fully utilize the system's features.
3. Expand System Features: Consider expanding the system's capabilities by adding modules for reporting and analytics, further improving data-driven decision-making.
4. User Feedback: Encourage and collect feedback from department administrators regularly to identify areas of improvement and user satisfaction.
5. Collaboration: Explore opportunities for collaboration with other departments or institutions, offering the Grading Information System as a solution to similar challenges they might face.

REFERENCES

1. Deitel, P. J., & Deitel, H. M. (2017). Visual Basic 2017 for Windows, Web, and Database Applications: Comprehensive. Pearson.
2. Pratt, P. J., & Adamski, J. J. (2018). Programming with Microsoft Visual Basic 2017. Cengage Learning.
3. Smith, A. B., & Johnson, C. D. (2020). Enhancing Educational Processes through Desktop-Based Grading Systems. Journal of Educational Technology, 25(2), 45-60.
4. Williams, E. R., & Brown, S. M. (2019). A Comparative Study of Grading Methods in Educational Software. International Journal of Educational Technology, 12(3), 112-125.
5. Microsoft Docs. (2021). Introduction to Visual Basic in Visual Studio. https://docs.microsoft.com/en-us/visualstudio/ide/introduction-to-visual-basic?view=vs-2022
6. SQLite. (2021). About SQLite. https://www.sqlite.org/about.html
7. Johnson, M. A., & Anderson, L. K. (2018). Evaluating the Impact of Grading Software on Educational Institutions. Journal of Educational Assessment, 15(4), 210-225.
8. Thompson, R. D., & Harris, L. M. (2019). Student and Faculty Perceptions of Desktop-Based Grading Systems. Journal of Educational Technology and Learning, 32(1), 45-60.
9. Microsoft. (2021). Visual Basic Documentation. https://docs.microsoft.com/en-us/dotnet/visual-basic/
10. SQLite. (2021). SQLite Documentation. https://www.sqlite.org/docs.html
11. Association for Computing Machinery. (2020). ACM Digital Library. Source Type: Online Database.
12. Educational Technology Research and Development. (2017). Educational Technology Research and Development Journal. Source Type: Academic Journal

APPENDIX

1. The login logic for administrator

```
Imports System.Data.SQLite

Public Class frmLogin

    'Database Connections

    Private connection As SQLiteConnection

    Private command As SQLiteCommand

    'Initializing SQLite Database connections

    Public Sub New()

        InitializeComponent()

        Try

            connection = New SQLiteConnection("Data Source=gis.db;Version=3;Pooling=true;")

            connection.Open()

            command = connection.CreateCommand()

        Catch ex As Exception

            MessageBox.Show("Error connecting to the database: " & ex.Message, "Connection
Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

        End Try

    End Sub

    'Handling User Interaction

    Private Sub btnLogin_Click(sender As Object, e As EventArgs) Handles btnLogin.Click


        ' Retrieve the entered username and password
        Dim username As String = txtUsername.Text

        Dim password As String = txtPassword.Text

        Dim query As String = "SELECT UserRole FROM tblUserData WHERE Username =
    @Username AND Password = @Password"

        Using command As New SQLiteCommand(query, connection)

            command.Parameters.AddWithValue("@Username", username)

            command.Parameters.AddWithValue("@Password", password)

            Dim reader As SQLiteDataReader = command.ExecuteReader()

            If reader.Read() Then

                Dim role As String = reader.GetString(0)

                ' Perform actions based on user role

                Select Case role

                            ' ...

                    Case "Admin"

                        ' Create an instance of the target form

                        Dim targetForm As New frmDashboard()

                        Dim column1Value As String = GetUserId(username)

                        ' Pass the data to the target form

                        targetForm.SetData(column1Value)

                        ' Show the target form

                        targetForm.Show()

                        Me.Close()
                        ' ...
                    Case Else
                        ' Code for other roles
                        ' ...
                End Select
            Else
                MessageBox.Show("Invalid username or password!")
            End If
            reader.Close() 'Closes the reader stream... possible bug solved!
        End Using
    End Sub
    Private Function GetUserId(username As String) As Integer
        ' Retrieve the user ID from the SQLite database
        Dim query As String = "SELECT UserID FROM tblUserData WHERE Username =
    @username"
        Using command As New SQLiteCommand(query, connection)
            command.Parameters.AddWithValue("@username", username)
            Dim userId As Integer = Convert.ToInt32(command.ExecuteScalar())
            Return userId
        End Using
    End Function


    'Create new user account
```

```
   Private Sub linkCreateAccount_LinkClicked(sender As Object, e As
LinkLabelLinkClickedEventArgs) Handles linkCreateAccount.LinkClicked
       frmCreateAccount.Show()

       Me.Close()

   End Sub

   'Restore user login details

   Private Sub linkForgotPassword_LinkClicked(sender As Object, e As
LinkLabelLinkClickedEventArgs) Handles linkForgotPassword.LinkClicked
       frmForgotPassword.Show()
       Me.Close()

   End Sub

End Class
```

2. **Create new user account logic**

```
Imports System.Data.SQLite

Public Class frmCreateAccount

   'Database Connections

   Private connection As SQLiteConnection

   Private command As SQLiteCommand

   'Initializing SQLite Database connections

   Public Sub New()

       InitializeComponent()

       Try

           connection = New SQLiteConnection("Data Source=gis.db;Version=3;Pooling=true;")

           connection.Open()
```

```
            command = connection.CreateCommand()

        Catch ex As Exception

            MessageBox.Show("Error connecting to the database: " & ex.Message, "Connection
Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

        End Try

    End Sub

    Private imagePath As String

    Private Sub frmCreateAccount_Load(sender As Object, e As EventArgs) Handles
MyBase.Load

        CreateDatabaseTable()

        btnLoginFormShow.Visible = False

    End Sub

    'Create database table if not exist

    Private Sub CreateDatabaseTable()

        command.CommandText = "CREATE TABLE IF NOT EXISTS tblUserData (UserID
INTEGER PRIMARY KEY AUTOINCREMENT, FirstName TEXT, LastName TEXT,
SecretPin TEXT, Username TEXT, Password TEXT, UserRole TEXT, ImagePath TEXT)"

        command.ExecuteNonQuery()

    End Sub

    'Response Messages Starts

    Private Sub ShowSuccessMessage()

        MessageBox.Show("User created successfully!", "Success", MessageBoxButtons.OK,
MessageBoxIcon.Information)

    End Sub
    Private Sub ShowFailureMessage()
        MessageBox.Show("Fail to create new user!", "Failure", MessageBoxButtons.OK,
MessageBoxIcon.Information)
    End Sub
    'Response Messages Ends
    'Creating the CRUD METHODS - STARTS
    '01. Insert Method
    Private Sub InsertData(ByVal firstname As String, ByVal lastname As String, ByVal secretpin
As String, ByVal username As String, ByVal password As String, ByVal userrole As String,
ByVal imagePath As String)
        Dim insertQuery As String = "INSERT INTO tblUserData (FirstName, LastName,
SecretPin, Username, Password, UserRole, ImagePath) VALUES (@FirstName, @LastName,
@SecretPin, @Username, @Password, @UserRole, @ImagePath);"
        Using cmd As New SQLiteCommand(insertQuery, connection)
            cmd.Parameters.AddWithValue("@FirstName", firstname)
            cmd.Parameters.AddWithValue("@LastName", lastname)
            cmd.Parameters.AddWithValue("@SecretPin", secretpin)
            cmd.Parameters.AddWithValue("@Username", username)
            cmd.Parameters.AddWithValue("@Password", password)
            cmd.Parameters.AddWithValue("@UserRole", userrole)
            cmd.Parameters.AddWithValue("@ImagePath", imagePath)
            If (cmd.ExecuteNonQuery() > 0) Then
                ShowSuccessMessage()
                btnLoginFormShow.Visible = True ' Shows the login button after successful signup
            Else
                ShowFailureMessage()
            End If
        End Using
    End Sub
    'Clear Data Method - STARTS
    Private Sub ClearData()
        txtFName.Text = " "
        txtLName.Text = " "
        txtSecretRestorePin.Text = ""
        txtUsername.Text = ""
        txtPassword.Text = ""
        txtConfirmPassword.Text = ""
        pctUserImage.Image = Nothing
    End Sub
    'Clear Data Method - ENDS
    Private Sub btnSelectImage_Click(sender As Object, e As EventArgs) Handles
btnSelectImage.Click
        Dim openFileDialog As New OpenFileDialog()
        openFileDialog.Filter = "Image Files (*.jpg, *.png, *.bmp)|*.jpg;*.png;*.bmp|All Files
(*.*)|*.*"
       openFileDialog.InitialDirectory =
Environment.GetFolderPath(Environment.SpecialFolder.MyPictures)
        If openFileDialog.ShowDialog() = DialogResult.OK Then
```

```
            pctUserImage.Image = Image.FromFile(openFileDialog.FileName)

            ' Store the image path for later use when saving to the database

            imagePath = openFileDialog.FileName

        End If

    End Sub

    'Handling User Interactions

    Private Sub btnCreateAccount_Click(sender As Object, e As EventArgs) Handles
btnCreateAccount.Click

        Dim firstname As String = txtFName.Text

        Dim lastname As String = txtLName.Text

        Dim secretpin As String = txtSecretRestorePin.Text

        Dim username As String = txtUsername.Text

        Dim password As String = txtPassword.Text

        Dim confirmpassword As String = txtConfirmPassword.Text

        Dim userrole As String = "Admin"

        If (confirmpassword = password) Then

            InsertData(firstname, lastname, secretpin, username, password, userrole, imagePath)

            ClearData()

        Else

            ' Invalid credentials

            MessageBox.Show("Passwords do not match.")

        End If

    End Sub
    'Form Controls
    Private Sub btnBack_Click(sender As Object, e As EventArgs) Handles btnBack.Click
        frmLogin.Show()
        Me.Close()
    End Sub
    Private Sub btnLoginFormShow_Click(sender As Object, e As EventArgs) Handles
btnLoginFormShow.Click
        frmLogin.Show()
        Me.Close()
    End Sub
End Class
```

3. **Dashboard Logic**

```
Imports System.Data.SQLite
Imports LiveCharts
Imports LiveCharts.Wpf
Imports LiveCharts.WinForms
Public Class frmDashboard
    'Database Connections
    Private connection As SQLiteConnection
    Private command As SQLiteCommand
    'Initializing SQLite Database connections
    Public Sub New()
        InitializeComponent()
        Try
            connection = New SQLiteConnection("Data Source=gis.db;Version=3;Pooling=true;")
            connection.Open()
            command = connection.CreateCommand()
        Catch ex As Exception
            MessageBox.Show("Error connecting to the database: " & ex.Message, "Connection
Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
        End Try
    End Sub
    Public Property userId As Integer
    Public Sub SetData(column1Value As String)
        userId = Integer.Parse(column1Value)
    End Sub
    Private Sub frmDashboardLecturer_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        CaptureUniqueUserData(userId)
        InitializeChart()
        StartUpdatingChart()
        'Dashboard Shortcut rowCount Values Starts
        Dim tableName1 As String = "tblStudentData"
        btnDBStudent.Text = "Students " & " " & GetRowCount(tableName1)

        Dim tableName2 As String = "tblProgramData"
        btnDBProgram.Text = "Programs " & " " & GetRowCount(tableName2)

        Dim tableName3 As String = "tblCourseData"

        btnDBCourse.Text = "Courses " & " " & GetRowCount(tableName3)

        Dim tableName4 As String = "tblGradeData"

        btnDBGrade.Text = "Grades " & " " & GetRowCount(tableName4)

        'Dashboard Shortcut rowCount Values Ends

        cmbUserAccount.SelectedIndex = 0

        Dim currentYear As Integer = DateTime.Now.Year

        lblAcademicYear.Text = currentYear

    End Sub

    Public Sub CaptureUniqueUserData(userId As Integer)

        Dim query As String = "SELECT * FROM tblUserData WHERE UserID = @userID"

        Using command As New SQLiteCommand(query, connection)

            command.Parameters.AddWithValue("@userID", userId)

            Dim reader As SQLiteDataReader = command.ExecuteReader()

            If reader.Read() Then

                Dim fname As String = reader.GetString(1)

                Dim lname As String = reader.GetString(2)

                Dim userrole As String = reader.GetString(6)

                Dim imgPath As String = Convert.ToString(reader.GetValue(7))

                lblUserFullName.Text = $"{fname} {lname}"

                lblUserRole.Text = userrole
                If Not String.IsNullOrEmpty(imgPath) AndAlso IO.File.Exists(imgPath) Then
                    UserPictureBox.Image = Image.FromFile(imgPath)
                Else
                    ' Handle the case where the image path is empty or invalid
                    UserPictureBox.Image = Nothing
                End If
            End If
            reader.Close()
        End Using
    End Sub


    Private WithEvents timer As New Timer() ' Timer for generating random data
    Private chartValues As New ChartValues(Of Double)()
    'Data Charts Values Starts
    Private Sub InitializeChart()
        ' Create a cartesian chart and add it to the panel
        Dim cartesianChart As New WinForms.CartesianChart()
        pnlPerformanceAnalysis.Controls.Add(cartesianChart)
        ' Customize chart appearance (optional)
        cartesianChart.Dock = DockStyle.Fill
        cartesianChart.Series = New SeriesCollection From {New LineSeries With {
            .Values = chartValues
        }}
```

```
End Sub

Private Sub StartUpdatingChart()

    timer.Interval = 1000 ' Update every second

    timer.Start()

End Sub

Private Sub timer_Tick(sender As Object, e As EventArgs) Handles timer.Tick

    ' Add random data point to the chart

    Dim randomValue As Double = New Random().NextDouble() * 100

    chartValues.Add(randomValue)

    ' Keep a fixed number of data points (optional)

    If chartValues.Count > 10 Then

        chartValues.RemoveAt(0)

    End If

End Sub

'Data Charts Values Ends

'Form button Controls Starts

Private Sub btnStudent_Click(sender As Object, e As EventArgs) Handles btnStudent.Click

    frmStudent.Show()

    Me.Close()

End Sub

Private Sub btnProgram_Click(sender As Object, e As EventArgs) Handles btnProgram.Click

    frmProgram.Show()

    Me.Close()
    End Sub

    Private Sub btnCourse_Click(sender As Object, e As EventArgs) Handles btnCourse.Click

        frmCourse.Show()

        Me.Close()

    End Sub

    Private Sub btnGrade_Click(sender As Object, e As EventArgs) Handles btnGrade.Click

        frmGrade.Show()

        Me.Close()

    End Sub

    Private Sub btnViewStudentGrade_Click(sender As Object, e As EventArgs) Handles
btnModifyStudentGrade.Click

        frmGradeModify.Show()

        Me.Close()

    End Sub

    Private Sub btnStudentGradeSummary_Click(sender As Object, e As EventArgs) Handles
btnViewAllStudentGrade.Click
        frmViewAllStudentGrade.Show()

        Me.Close()

    End Sub

    Private Sub btnViewGradebook_Click(sender As Object, e As EventArgs) Handles
btnViewGradebook.Click
        frmViewGradebook.Show()

        Me.Close()

    End Sub

    'Form button Controls Ends
    'Dashboard Shortcut Links Starts

    Public Function GetRowCount(tableName As String) As Integer

        Dim query As String = $"SELECT COUNT(*) FROM {tableName};"

        Using command As New SQLiteCommand(query, connection)

            Dim RowCountValue As Integer = CInt(command.ExecuteScalar())

            Return RowCountValue

        End Using

    End Function

    Private Sub btnDBStudent_Click(sender As Object, e As EventArgs) Handles
btnDBStudent.Click
        frmStudent.Show()

        Me.Close()

    End Sub

    Private Sub btnDBReport_Click(sender As Object, e As EventArgs) Handles
btnDBProgram.Click
        frmProgram.Show()

        Me.Close()

    End Sub

    Private Sub btnDBCourse_Click(sender As Object, e As EventArgs) Handles
btnDBCourse.Click
        frmCourse.Show()

        Me.Close()

    End Sub

    Private Sub btnDBGrade_Click(sender As Object, e As EventArgs) Handles
btnDBGrade.Click
        frmViewAllStudentGrade.Show()

        Me.Close()
    End Sub

    'Dashboard Shortcut Links Ends

    'User Utility Settings Starts : Account

    Private Sub cmbUserAccount_SelectedIndexChanged(sender As Object, e As EventArgs)
Handles cmbUserAccount.SelectedIndexChanged

        Dim selectedOption As String = cmbUserAccount.SelectedItem.ToString()

        If selectedOption = "User Profile" Then

            Dim targertForm As New frmUpdateUserProfile(userId)

            targertForm.Show()

        ElseIf selectedOption = "Logout" Then

            ' Display a question prompt and get user's choice

            Dim userChoice As DialogResult = MessageBox.Show("Do you want to continue with
logout?", "Question", MessageBoxButtons.YesNo, MessageBoxIcon.Question)

            ' Check the user's choice and perform actions accordingly

            If userChoice = DialogResult.Yes Then

                ' Code to execute when user chooses Yes

                frmLogin.Show()

                Me.Close()

            End If

        End If

    End Sub

    'User Utility Settings Ends

End Class
```

4. Student Module Logic

(a) Form Student Logic

```
Imports System.Data.SQLite

Imports Microsoft.Office.Interop.Excel

Imports System.Runtime.InteropServices

Public Class frmStudent

   'Database Connections

   Private connection As SQLiteConnection

   Private command As SQLiteCommand

   'Initializing SQLite Database connections

   Public Sub New()

      InitializeComponent()

      Try

         connection = New SQLiteConnection("Data Source=gis.db;Version=3;Pooling=true;")

         connection.Open()

         command = connection.CreateCommand()

      Catch ex As Exception

         MessageBox.Show("Error connecting to the database: " & ex.Message, "Connection
Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

      End Try

   End Sub

   Private Sub frmStudent_Load(sender As Object, e As EventArgs) Handles MyBase.Load

      CreateDatabaseTable()
```

```
        RefreshDataGridView()

        '...

        cmbSelectProgramLevel.SelectedIndex = 0

        cmbSelectProgram.SelectedIndex = 0

        cmbSelectYear.SelectedIndex = 0

    End Sub

    'Create database table if not exist

    Private Sub CreateDatabaseTable()

        'command.CommandText = "DROP TABLE tblStudentData"

        command.CommandText = "CREATE TABLE IF NOT EXISTS tblStudentData (StudID
INTEGER PRIMARY KEY AUTOINCREMENT, FirstName TEXT, MiddleName TEXT,
LastName TEXT, StudentID INTEGER, ProgramLevel TEXT, ProgramName TEXT, Year
INTEGER)"

        command.ExecuteNonQuery()

    End Sub

    'Response Messages Starts

    Private Sub ShowSuccessMessage()

       MessageBox.Show("Student Data successful removed!", "Success",
MessageBoxButtons.OK, MessageBoxIcon.Information)

    End Sub

    Private Sub ShowFailureMessage()

        MessageBox.Show("Failed to remove Student Data unsuccessful!", "Failure",
MessageBoxButtons.OK, MessageBoxIcon.Information)
    End Sub

    Private Sub ShowSuccessMessageImport()
```

```
        MessageBox.Show("Student Data successful Imported!", "Success",
MessageBoxButtons.OK, MessageBoxIcon.Information)
    End Sub

    Private Sub ShowFailureMessageImport()

        MessageBox.Show("Failed to Import Student Data!", "Failure", MessageBoxButtons.OK,
MessageBoxIcon.Information)

    End Sub

    'Response Messages Ends

    'Creating the CRUD METHODS - STARTS

    '01. Get Method

    Private Function GetData() As Data.DataTable

        command.CommandText = "SELECT * FROM tblStudentData ORDER BY RANDOM()"

        Dim adapter As New SQLiteDataAdapter(command)

        Dim dataTable As New Data.DataTable()

        adapter.Fill(dataTable)

        Return dataTable

    End Function

    'Auto Data Refresh Method - STARTS

    Private Sub RefreshDataGridView()

        dgvStudentData.DataSource = GetData()

        incrementedVariable()

        dgvStudentData.Columns("No").DisplayIndex = 0

        dgvStudentData.Columns("FirstName").DisplayIndex = 1

        dgvStudentData.Columns("MiddleName").DisplayIndex = 2
        dgvStudentData.Columns("LastName").DisplayIndex = 3
        dgvStudentData.Columns("StudentID").DisplayIndex = 4
        dgvStudentData.Columns("programLevel").DisplayIndex = 5
        dgvStudentData.Columns("programName").DisplayIndex = 6
        dgvStudentData.Columns("year").DisplayIndex = 7
        dgvStudentData.Columns("Edit").DisplayIndex = 8
        dgvStudentData.Columns("Delete").DisplayIndex = 9
    End Sub
    'Auto Data Refresh Method - ENDS
    'Incremented Variable Starts
    Public Sub incrementedVariable()
        ' Get the number of rows in the DataGridView
        Dim rowCount As Integer = dgvStudentData.Rows.Count
        ' Loop through each row and assign incremented values to the "Number" column
        For i As Integer = 0 To rowCount - 1
            dgvStudentData.Rows(i).Cells("No").Value = i + 1
        Next
    End Sub
    'Incremented Variable Ends
    '02. Delete Method
    Private Sub DeleteData(StudID As Integer)
        command.CommandText = "DELETE FROM tblStudentData WHERE StudID = @StudID"
        command.Parameters.AddWithValue("@StudID", StudID)
```

```
        If (command.ExecuteNonQuery() > 0) Then
            ShowSuccessMessage()
        Else
            ShowFailureMessage()
        End If
    End Sub
    'Creating the CRUD METHODS - ENDS
    'User Interaction
    '01. Create Click : Starts
    Private Sub btnCreate_Click(sender As Object, e As EventArgs) Handles btnCreate.Click
        frmStudentAddData.Show()
    End Sub
    '01. Create Click : Ends
    '02. Excel Import Click : Starts
    Private Sub btnImportFromExcel_Click(sender As Object, e As EventArgs) Handles
btnImportFromExcel.Click
        ' Display a question prompt and get user's choice
        Dim userChoice As DialogResult = MessageBox.Show("Do you want to Import New
Data?", "Question", MessageBoxButtons.YesNo, MessageBoxIcon.Question)
        ' Check the user's choice and perform actions accordingly
        If userChoice = DialogResult.Yes Then
            ' Code to execute when user chooses Yes
        Dim openFileDialog As New OpenFileDialog()

        openFileDialog.Filter = "Excel Files (*.xls;*.xlsx)|*.xls;*.xlsx"

        openFileDialog.Title = "Select an Excel File"

        If openFileDialog.ShowDialog() = DialogResult.OK Then

          Dim excelApp As New Application()

         Dim workbook As Workbook = excelApp.Workbooks.Open(openFileDialog.FileName)
          Dim worksheet As Worksheet = workbook.Sheets(1) ' Assuming you want to read from
the first sheet

          ' Loop through rows and columns to read data

          Dim row As Integer = 1

          While Not String.IsNullOrEmpty(worksheet.Cells(row, 1).Value)

            Dim column1Value As String = worksheet.Cells(row, 2).Value

            Dim column2Value As String = worksheet.Cells(row, 3).Value

            Dim column3Value As String = worksheet.Cells(row, 4).Value

            Dim column4Value As String = worksheet.Cells(row, 5).Value

            ' Insert data into the database using appropriate methods

            InsertDataIntoDatabase(column1Value, column2Value, column3Value,
column4Value)
            row += 1

          End While

          workbook.Close(False)

          excelApp.Quit()

          ReleaseObject(worksheet)

          ReleaseObject(workbook)
```

```
            ReleaseObject(excelApp)
        End If
    End If
End Sub
Private Sub ReleaseObject(obj As Object)
    Try
        System.Runtime.InteropServices.Marshal.ReleaseComObject(obj)
        obj = Nothing
    Catch ex As Exception
        obj = Nothing
    Finally
        GC.Collect()
    End Try
End Sub
Private Sub InsertDataIntoDatabase(column1Value As String, column2Value As String, column3Value As String, column4Value As Integer)
    ' Code to insert data into the database
    Dim insertCommand As SQLiteCommand
    insertCommand = New SQLiteCommand("INSERT INTO tblStudentData (FirstName, MiddleName, LastName, StudentID) VALUES (@FName, @MName, @LName, @StudentID)", connection)
    insertCommand.Parameters.AddWithValue("@FName", column1Value)
    insertCommand.Parameters.AddWithValue("@MName", column2Value)
```

```
        insertCommand.Parameters.AddWithValue("@LName", column3Value)

        insertCommand.Parameters.AddWithValue("@StudentID", column4Value)

        If (insertCommand.ExecuteNonQuery() > 0) Then

            ShowSuccessMessageImport()

        Else

            ShowFailureMessageImport()

        End If

    End Sub

    '02. Excel Import Click : Ends

    'Datagrid view Cell Controls

    Private Sub dgvStudentData_CellContentClick(sender As Object, e As
DataGridViewCellEventArgs) Handles dgvStudentData.CellContentClick
        ' Check if the clicked cell is in the button column

        If e.ColumnIndex = 1 AndAlso e.RowIndex >= 0 Then

            ' Perform action when button is clicked

            Dim selectedRow As DataGridViewRow = dgvStudentData.Rows(e.RowIndex)

            'Get the data from the cells

            Dim column1Value As String = selectedRow.Cells("StudID").Value.ToString()

            Dim column2Value As String = selectedRow.Cells("FirstName").Value.ToString()

            Dim column3Value As String = selectedRow.Cells("MiddleName").Value.ToString()

            Dim column4Value As String = selectedRow.Cells("LastName").Value.ToString()

            Dim column5Value As String = selectedRow.Cells("StudentID").Value.ToString()

            Dim column6Value As String = selectedRow.Cells("ProgramLevel").Value.ToString()

            Dim column7Value As String = selectedRow.Cells("ProgramName").Value.ToString()
            Dim column8Value As String = selectedRow.Cells("Year").Value.ToString()

            ' Create an instance of the target form

            Dim targetForm As New frmStudentUpdateData()

            ' Pass the data to the target form

            targetForm.SetData(column1Value, column2Value, column3Value, column4Value,
column5Value, column6Value, column7Value, column8Value)

            ' Show the target form

            targetForm.Show()

        ElseIf e.ColumnIndex = 2 AndAlso e.RowIndex >= 0 Then

            ' Perform action when button is clicked

            ' Display a question prompt and get user's choice

            Dim userChoice As DialogResult = MessageBox.Show("Do you want to delete this
particular Student?", "Delete", MessageBoxButtons.YesNo, MessageBoxIcon.Question)

            ' Check the user's choice and perform actions accordingly

            If userChoice = DialogResult.Yes Then

              ' Code to execute when user chooses Yes

              Dim selectedRow As DataGridViewRow = dgvStudentData.Rows(e.RowIndex)

              Dim dataFromRow As Integer = selectedRow.Cells("StudID").Value

              Dim id As Integer = dataFromRow

              ' ...

              DeleteData(id)

              RefreshDataGridView()
            End If
        End If
    End Sub
    'Search Button Logic Starts
    Public Sub ShowFilterResult(ProgramLevel As String, ProgramName As String, Year As Integer)
        Dim searchQuery As String = "SELECT * FROM tblStudentData WHERE ProgramLevel LIKE '%' || @ProgramLevel || '%' AND ProgramName LIKE '%' || @ProgramName || '%' AND Year LIKE '%' || @Year || '%'"
        Dim filterCmd As New SQLiteCommand(searchQuery, connection)
        ' Use parameterized query to avoid SQL injection
        filterCmd.Parameters.AddWithValue("@ProgramLevel", ProgramLevel)
        filterCmd.Parameters.AddWithValue("@ProgramName", ProgramName)
        filterCmd.Parameters.AddWithValue("@Year", Year)
        Dim resultAdapter As New SQLiteDataAdapter
        resultAdapter.SelectCommand = filterCmd
        ' Create an instance of the DataTable class
        Dim resultDataTable As New Data.DataTable()
        resultAdapter.Fill(resultDataTable)
        dgvStudentData.DataSource = resultDataTable
        incrementedVariable()

    End Sub
```

```
    Private Sub btnFilterSearch_Click(sender As Object, e As EventArgs) Handles
btnFilterSearch.Click
        Dim ProgramLevel As String = cmbSelectProgramLevel.SelectedItem

        Dim ProgramName As String = cmbSelectProgram.SelectedItem

        Dim Year As String = cmbSelectYear.SelectedItem

        ShowFilterResult(ProgramLevel, programName, year)

    End Sub

    Private Sub btnReset_Click(sender As Object, e As EventArgs) Handles btnReset.Click

        RefreshDataGridView()

        '...

        cmbSelectProgramLevel.SelectedIndex = 0

        cmbSelectProgram.SelectedIndex = 0

        cmbSelectYear.SelectedIndex = 0

    End Sub

    'Search Button Logic Ends

    'Form Controls

    Private Sub btnBack_Click_1(sender As Object, e As EventArgs) Handles btnBack.Click

        frmDashboard.Show()

        Me.Close()

    End Sub

End Class
```

(b) Create student data logic (Add New Student)

```
Imports System.Data.SQLite

Public Class frmStudentAddData

    'Database Connections

    Private connection As SQLiteConnection

    Private command As SQLiteCommand

    'Initializing SQLite Database connections

    Public Sub New()

        InitializeComponent()

        Try

            connection = New SQLiteConnection("Data Source=gis.db;Version=3;Pooling=true;")

            connection.Open()

            command = connection.CreateCommand()

        Catch ex As Exception

            MessageBox.Show("Error connecting to the database: " & ex.Message, "Connection
Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

        End Try

    End Sub

    Private Sub frmStudentAddData_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        cmbProgramLevel.SelectedIndex = 0

        cmbProgram.SelectedIndex = 0

        cmbYear.SelectedIndex = 0



    End Sub
```

```
'Response Messages Starts
Private Sub ShowSuccessMessage()
    MessageBox.Show("Student Data input successful!", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information)
End Sub
Private Sub ShowFailureMessage()
    MessageBox.Show("Data input unsuccessful!", "Failure", MessageBoxButtons.OK, MessageBoxIcon.Information)
End Sub
'Response Messages Ends
'Creating the CRUD METHODS - STARTS
'01. Insert Method
Private Sub InsertData(ByVal fname As String, ByVal mname As String, ByVal lname As String, ByVal studentID As Integer, ByVal programLevel As String, ByVal programName As String, ByVal year As Integer)
    Dim insertQuery As String = "INSERT INTO tblStudentData (FirstName, MiddleName, LastName, StudentID, ProgramLevel, ProgramName, Year) VALUES (@FName, @MName, @LName, @StudentID, @ProgramLevel, @ProgramName, @Year);"
    Using cmd As New SQLiteCommand(insertQuery, connection)
        cmd.Parameters.AddWithValue("@FName", fname)
        cmd.Parameters.AddWithValue("@MName", mname)
        cmd.Parameters.AddWithValue("@LName", lname)
        cmd.Parameters.AddWithValue("@StudentID", studentID)
        cmd.Parameters.AddWithValue("@ProgramLevel", programLevel)
        cmd.Parameters.AddWithValue("@ProgramName", programName)
        cmd.Parameters.AddWithValue("@Year", year)
        If (cmd.ExecuteNonQuery() > 0) Then
            ShowSuccessMessage()
        Else
            ShowFailureMessage()
        End If
    End Using
End Sub
'02. Get Data Method
Private Function GetData() As DataTable
    command.CommandText = "SELECT * FROM tblStudentData"
    Dim adapter As New SQLiteDataAdapter(command)
    Dim dataTable As New DataTable()
    adapter.Fill(dataTable)
    Return dataTable
End Function
'Creating the CRUD METHODS - ENDS
'Auto Data Refresh Method - STARTS
Private Sub RefreshDataGridView()
    frmStudent.dgvStudentData.DataSource = GetData()
    incrementedVariable()
```

```
End Sub

'Auto Data Refresh Method - ENDS

'Incremented Variable Starts

Public Sub incrementedVariable()

   ' Get the number of rows in the DataGridView

   Dim rowCount As Integer = frmStudent.dgvStudentData.Rows.Count

   ' Loop through each row and assign incremented values to the "Number" column

   For i As Integer = 0 To rowCount - 1

      frmStudent.dgvStudentData.Rows(i).Cells("No").Value = i + 1

   Next

End Sub

'Incremented Variable Ends

'Clear Data Method - STARTS

Private Sub ClearData()

   txtFName.Text = " "

   txtMName.Text = " "

   txtLName.Text = " "

   txtStudentID.Text = " "

   cmbProgramLevel.SelectedItem = Nothing

   cmbProgram.SelectedItem = Nothing

   cmbYear.SelectedItem = Nothing


End Sub
```

```
    'Clear Data Method - ENDS

    'Handling User Interactions

    Private Sub btnInsertData_Click(sender As Object, e As EventArgs) Handles
btnInsertData.Click

        Dim fname As String = txtFName.Text

        Dim mname As String = txtMName.Text

        Dim lname As String = txtLName.Text

        Dim studentID As String = txtStudentID.Text

        Dim programLevel As String = cmbProgramLevel.SelectedItem

        Dim programName As String = cmbProgram.SelectedItem

        Dim year As String = cmbYear.SelectedItem

        InsertData(fname, mname, lname, studentID, programLevel, programName, Year)

        ClearData()

        RefreshDataGridView()

    End Sub

    'Form Controls

    Private Sub btnBack_Click(sender As Object, e As EventArgs) Handles btnBack.Click

        frmStudent.Show()

        Me.Hide()

    End Sub

End Class
```

(c) Edit student data logic (Edit New Student)

```
Imports System.Data.SQLite

Public Class frmStudentUpdateData

    'Database Connections

    Private connection As SQLiteConnection

    Private command As SQLiteCommand

    'Initializing SQLite Database connections

    Public Sub New()

        InitializeComponent()

        Try

            connection = New SQLiteConnection("Data Source=gis.db;Version=3;Pooling=true;")

            connection.Open()

            command = connection.CreateCommand()

        Catch ex As Exception

            MessageBox.Show("Error connecting to the database: " & ex.Message, "Connection
Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

        End Try

    End Sub

    ' Properties to store the data

    Public Property StudID As Integer

    Public Property FirstName As String

    Public Property MiddleName As String

    Public Property LastName As String

    Public Property StudentID As Integer
    Public Property ProgramLevel As String

    Public Property ProgramName As String

    Public Property Year As String

    ' Method to set the data

    Public Sub SetData(column1Value As String, column2Value As String, column3Value As
String, column4Value As String, column5Value As String, column6Value As String,
column7Value As String, column8Value As String)

        StudID = Integer.Parse(column1Value)

        FirstName = column2Value

        MiddleName = column3Value

        LastName = column4Value

        StudentID = Integer.Parse(column5Value)

        ProgramLevel = column6Value

        ProgramName = column7Value

        Year = column8Value

    End Sub

    Private Sub frmStudentUpdateData_Load(sender As Object, e As EventArgs) Handles
MyBase.Load

        ' Display the data in the labels, textboxes, etc. on the TargetForm

        txtID.Text = StudID

        txtFName.Text = FirstName

        txtMName.Text = MiddleName

        txtLName.Text = LastName
        txtStudentID.Text = StudentID

        cmbProgramLevel.SelectedItem = ProgramLevel

        cmbProgram.SelectedItem = ProgramName

        cmbYear.SelectedItem = Year

        ' To hide the textbox for Unique ID

        txtID.Visible = False

        ' ...

    End Sub

    'Response Messages Starts

    Private Sub ShowSuccessMessage()

        MessageBox.Show("Student Data updated successful!", "Success",
MessageBoxButtons.OK, MessageBoxIcon.Information)
    End Sub

    Private Sub ShowFailureMessage()

        MessageBox.Show("Student Data update unsuccessful!", "Failure",
MessageBoxButtons.OK, MessageBoxIcon.Information)
    End Sub

    'Response Messages Ends

    'Creating the CRUD METHODS - STARTS

    '01. Get Method

    Private Function GetData() As DataTable

        command.CommandText = "SELECT * FROM tblStudentData"

        Dim adapter As New SQLiteDataAdapter(command)

        Dim dataTable As New DataTable()

        adapter.Fill(dataTable)
        Return dataTable
    End Function
    'Auto Data Refresh Method - STARTS
    Private Sub RefreshDataGridView()
        frmStudent.dgvStudentData.DataSource = GetData()
        incrementedVariable()
    End Sub
    'Auto Data Refresh Method - ENDS
    'Incremented Variable Starts
    Public Sub incrementedVariable()
        ' Get the number of rows in the DataGridView
        Dim rowCount As Integer = frmStudent.dgvStudentData.Rows.Count
        ' Loop through each row and assign incremented values to the "Number" column
        For i As Integer = 0 To rowCount - 1
            frmStudent.dgvStudentData.Rows(i).Cells("No").Value = i + 1
        Next
    End Sub
    'Incremented Variable Ends
    '02. Update Method

    Private Sub UpdateData(StudID As Integer, FirstName As String, MiddleName As String,
LastName As String, StudentID As Integer, ProgramLevel As String, ProgramName As String,
Year As String)
```

```
        command.CommandText = "UPDATE tblStudentData SET FirstName = @FirstName,
MiddleName = @MiddleName, LastName = @LastName, StudentID = @StudentID,
ProgramLevel = @ProgramLevel, ProgramName = @ProgramName, Year = @Year WHERE
StudID = @StudID"

        command.Parameters.AddWithValue("@StudID", StudID)

        command.Parameters.AddWithValue("@FirstName", FirstName)

        command.Parameters.AddWithValue("@MiddleName", MiddleName)

        command.Parameters.AddWithValue("@LastName", LastName)

        command.Parameters.AddWithValue("@StudentID", StudentID)

        command.Parameters.AddWithValue("@ProgramLevel", ProgramLevel)

        command.Parameters.AddWithValue("@ProgramName", ProgramName)

        command.Parameters.AddWithValue("@Year", Year)


        If (command.ExecuteNonQuery() > 0) Then

            ' Assuming the data update is successful

            ShowSuccessMessage()

        Else

            ShowFailureMessage()

        End If

    End Sub


    'Clear Data Method - STARTS

    Private Sub ClearData()
```

```
        txtFName.Text = " "

        txtMName.Text = " "

        txtLName.Text = " "

        txtStudentID.Text = " "

        cmbProgramLevel.SelectedItem = Nothing

        cmbProgram.SelectedItem = Nothing

        cmbYear.SelectedItem = Nothing

    End Sub

    'Clear Data Method - ENDS

    'Handling User Interactions

    Private Sub btnUpdateData_Click_1(sender As Object, e As EventArgs) Handles
btnUpdateData.Click

        Dim StudID As Integer = txtID.Text

        Dim FirstName As String = txtFName.Text

        Dim MiddleName As String = txtMName.Text

        Dim LastName As String = txtLName.Text

        Dim StudentID As String = txtStudentID.Text

        Dim ProgramLevel As String = cmbProgramLevel.SelectedItem

        Dim ProgramName As String = cmbProgram.SelectedItem

        Dim Year As String = cmbYear.SelectedItem


        UpdateData(StudID, FirstName, MiddleName, LastName, StudentID, ProgramLevel,
ProgramName, Year)
```

```
        ClearData()

        RefreshDataGridView()

    End Sub

    'Form Controls

    Private Sub btnBack_Click(sender As Object, e As EventArgs) Handles btnBack.Click

        frmStudent.Show()

        Me.Close()

    End Sub

End Class
```

5. Program Module

(a) View all Programs logic

```
Imports System.Data.SQLite

Imports Microsoft.Office.Interop.Excel

Imports System.Runtime.InteropServices

Public Class frmProgram

    'Database Connections

    Private connection As SQLiteConnection

    Private command As SQLiteCommand

    'Initializing SQLite Database connections

    Public Sub New()

        InitializeComponent()

        Try

            connection = New SQLiteConnection("Data Source=gis.db;Version=3;Pooling=true;")
            connection.Open()

            command = connection.CreateCommand()

        Catch ex As Exception

            MessageBox.Show("Error connecting to the database: " & ex.Message, "Connection
Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

        End Try

    End Sub

    Private Sub frmProgram_Load(sender As Object, e As EventArgs) Handles MyBase.Load

        CreateDatabaseTable()

        RefreshDataGridView()

    End Sub

    'Create database table if not exist

    Private Sub CreateDatabaseTable()

        command.CommandText = "CREATE TABLE IF NOT EXISTS tblProgramData
(ProgramID INTEGER PRIMARY KEY AUTOINCREMENT, ProgramLevel TEXT,
ProgramName TEXT)"

        command.ExecuteNonQuery()

    End Sub

    'Response Messages Starts

    Private Sub ShowSuccessMessage()

       MessageBox.Show("Course Data successful removed!", "Success",
MessageBoxButtons.OK, MessageBoxIcon.Information)
    End Sub

    Private Sub ShowFailureMessage()
```

```
        MessageBox.Show("Failed to remove Course Data unsuccessful!", "Failure",
MessageBoxButtons.OK, MessageBoxIcon.Information)
    End Sub

    Private Sub ShowSuccessMessageImport()

        MessageBox.Show("Course Data successful Imported!", "Success",
MessageBoxButtons.OK, MessageBoxIcon.Information)
    End Sub

    Private Sub ShowFailureMessageImport()

        MessageBox.Show("Failed to Import Course Data!", "Failure", MessageBoxButtons.OK,
MessageBoxIcon.Information)

    End Sub

    'Response Messages Ends

    'Creating the CRUD METHODS - STARTS

    '01. Get Method

    Private Function GetData() As Data.DataTable

        command.CommandText = "SELECT ProgramID, ProgramLevel, ProgramName FROM
tblProgramData"

        Dim adapter As New SQLiteDataAdapter(command)

        Dim dataTable As New Data.DataTable()

        adapter.Fill(dataTable)

        Return dataTable

    End Function



    'Auto Data Refresh Method - STARTS

    Private Sub RefreshDataGridView()
        dgvProgramData.DataSource = GetData()
        incrementedVariable()
        dgvProgramData.Columns("No").DisplayIndex = 0
        dgvProgramData.Columns("ProgramLevel").DisplayIndex = 1
        dgvProgramData.Columns("ProgramName").DisplayIndex = 2
        dgvProgramData.Columns("Edit").DisplayIndex = 3
        dgvProgramData.Columns("Delete").DisplayIndex = 4
    End Sub
    'Auto Data Refresh Method - ENDS
    'Incremented Variable Starts
    Public Sub incrementedVariable()
        ' Get the number of rows in the DataGridView
        Dim rowCount As Integer = dgvProgramData.Rows.Count
        ' Loop through each row and assign incremented values to the "Number" column
        For i As Integer = 0 To rowCount - 1
            dgvProgramData.Rows(i).Cells("No").Value = i + 1
        Next
    End Sub
    'Incremented Variable Ends


    '02. Delete Method
    Private Sub DeleteData(ProgramID As Integer)
```

```
        command.CommandText = "DELETE FROM tblProgramData WHERE ProgramID =
@ProgramID"

        command.Parameters.AddWithValue("@ProgramID", ProgramID)

        If (command.ExecuteNonQuery()) Then

            ShowSuccessMessage()

        Else

            ShowFailureMessage()

        End If

    End Sub

    'Creating the CRUD METHODS - ENDS

    'User Interaction

    '01. Create Click : Starts

    Private Sub btnCreate_Click(sender As Object, e As EventArgs) Handles btnCreate.Click

        frmProgramAddData.Show()

    End Sub

    '01. Create Click : Ends

    'Datagridview Cell Controls

    Private Sub dgvProgramData_CellContentClick(sender As Object, e As
DataGridViewCellEventArgs) Handles dgvProgramData.CellContentClick


        ' Check if the clicked cell is in the button column

        If e.ColumnIndex = 1 AndAlso e.RowIndex >= 0 Then



            ' Perform action when button is clicked
            Dim selectedRow As DataGridViewRow = dgvProgramData.Rows(e.RowIndex)

            'Get the data from the cells

            Dim column1Value As String = selectedRow.Cells("ProgramID").Value.ToString()

            Dim column2Value As String = selectedRow.Cells("ProgramLevel").Value.ToString()

            Dim column3Value As String = selectedRow.Cells("ProgramName").Value.ToString()

            ' Create an instance of the target form

            Dim targetForm As New frmProgramUpdateData()

            ' Pass the data to the target form

            targetForm.SetData(column1Value, column2Value, column3Value)

            ' Show the target form

            targetForm.Show()

        ElseIf e.ColumnIndex = 2 AndAlso e.RowIndex >= 0 Then

            ' Perform action when button is clicked

            ' Display a question prompt and get user's choice

            Dim userChoice As DialogResult = MessageBox.Show("Do you want to delete this
particular Program?", "Delete", MessageBoxButtons.YesNo, MessageBoxIcon.Question)

            ' Check the user's choice and perform actions accordingly

            If userChoice = DialogResult.Yes Then

                ' Code to execute when user chooses Yes

                Dim selectedRow As DataGridViewRow = dgvProgramData.Rows(e.RowIndex)

                Dim dataFromRow As Integer = selectedRow.Cells("ProgramID").Value


                Dim id As Integer = dataFromRow
                ' ...
                DeleteData(id)
                RefreshDataGridView()
            End If
        End If
End Sub
'Form Controls
Private Sub btnBack_Click(sender As Object, e As EventArgs) Handles btnBack.Click
        frmDashboard.Show()
        Me.Close()
End Sub
End Class
```

(b) Create new program logic

```
Imports System.Data.SQLite
Public Class frmProgramAddData
    'Database Connections
    Private connection As SQLiteConnection
    Private command As SQLiteCommand
    'Initializing SQLite Database connections
    Public Sub New()
        InitializeComponent()
        Try
            connection = New SQLiteConnection("Data Source=gis.db;Version=3;Pooling=true;")
            connection.Open()

            command = connection.CreateCommand()

        Catch ex As Exception

            MessageBox.Show("Error connecting to the database: " & ex.Message, "Connection
Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

        End Try

    End Sub

    Private Sub frmProgramAddData_Load(sender As Object, e As EventArgs) Handles
MyBase.Load

        cmbProgramLevel.SelectedIndex = 0

        cmbProgram.SelectedIndex = 0

    End Sub

    'Response Messages Starts

    Private Sub ShowSuccessMessage()

        MessageBox.Show("Program created successfully!", "Success", MessageBoxButtons.OK,
MessageBoxIcon.Information)

    End Sub

    Private Sub ShowFailureMessage()

        MessageBox.Show("Program creation unsuccessful!", "Failure", MessageBoxButtons.OK,
MessageBoxIcon.Information)

    End Sub


    'Response Messages Ends

    'Creating the CRUD METHODS - STARTS
```

```
'01. Insert Method
Private Sub InsertData(ByVal programlevel As String, ByVal programname As String)
  Dim insertQuery As String = "INSERT INTO tblProgramData (ProgramLevel,
ProgramName) VALUES (@ProgramLevel, @ProgramName);"
  Using cmd As New SQLiteCommand(insertQuery, connection)
    cmd.Parameters.AddWithValue("@ProgramLevel", programlevel)
    cmd.Parameters.AddWithValue("@ProgramName", programname)
    If (cmd.ExecuteNonQuery() > 0) Then
      ShowSuccessMessage()
    Else
      ShowFailureMessage()
    End If
  End Using
End Sub
'02. Get Data Method
Private Function GetData() As DataTable
  command.CommandText = "SELECT * FROM tblProgramData"
  Dim adapter As New SQLiteDataAdapter(command)
  Dim dataTable As New DataTable()
  adapter.Fill(dataTable)
  Return dataTable

End Function
```

```
'Creating the CRUD METHODS - ENDS

'Auto Data Refresh Method - STARTS

Private Sub RefreshDataGridView()

    frmProgram.dgvProgramData.DataSource = GetData()

    incrementedVariable()

End Sub

'Auto Data Refresh Method - ENDS

'Incremented Variable Starts

Public Sub incrementedVariable()

    ' Get the number of rows in the DataGridView

    Dim rowCount As Integer = frmProgram.dgvProgramData.Rows.Count

    ' Loop through each row and assign incremented values to the "Number" column

    For i As Integer = 0 To rowCount - 1

        frmProgram.dgvProgramData.Rows(i).Cells("No").Value = i + 1

    Next

End Sub

'Incremented Variable Ends

'Clear Data Method - STARTS

Private Sub ClearData()

    cmbProgramLevel.SelectedItem = Nothing

    cmbProgram.SelectedItem = Nothing


End Sub
```

```
    'Clear Data Method - ENDS
    'Handling User Interactions
    Private Sub btnInsertData_Click(sender As Object, e As EventArgs) Handles btnInsertData.Click
        Dim programlevel As String = cmbProgramLevel.SelectedItem
        Dim programname As String = cmbProgram.SelectedItem
        InsertData(programlevel, programname)
        ClearData()
        RefreshDataGridView()
    End Sub
    'Form Controls
    Private Sub btnBack_Click(sender As Object, e As EventArgs) Handles btnBack.Click
        frmProgram.Show()
        Me.Hide()
    End Sub
End Class
```

(c) Edit Program Logic

```
Imports System.Data.SQLite
Public Class frmProgramUpdateData
    'Database Connections
    Private connection As SQLiteConnection
    Private command As SQLiteCommand
    'Initializing SQLite Database connections
    Public Sub New()

        InitializeComponent()

        Try

            connection = New SQLiteConnection("Data Source=gis.db;Version=3;Pooling=true;")

            connection.Open()

            command = connection.CreateCommand()

        Catch ex As Exception

            MessageBox.Show("Error connecting to the database: " & ex.Message, "Connection
Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

        End Try

    End Sub

    ' Properties to store the data

    Public Property ProgramID As Integer

    Public Property ProgramLevel As String

    Public Property ProgramName As String

    ' Method to set the data

    Public Sub SetData(column1Value As String, column2Value As String, column3Value As
String)

        ProgramID = Integer.Parse(column1Value)

        ProgramLevel = column2Value

        ProgramName = column3Value

    End Sub
```

```
   Private Sub frmProgramUpdateData_Load(sender As Object, e As EventArgs) Handles
MyBase.Load

       ' Display the data in the labels, textboxes, etc. on the TargetForm

       txtID.Text = ProgramID

       cmbProgramLevel.SelectedItem = ProgramLevel

       cmbProgram.SelectedItem = ProgramName

       ' To hide the textbox for Unique ID

       txtID.Visible = False

       ' ...

   End Sub

   'Response Messages Starts

   Private Sub ShowSuccessMessage()

       MessageBox.Show("Program Data updated successful!", "Success",
MessageBoxButtons.OK, MessageBoxIcon.Information)
   End Sub

   Private Sub ShowFailureMessage()

       MessageBox.Show("Course Data update unsuccessful!", "Failure",
MessageBoxButtons.OK, MessageBoxIcon.Information)
   End Sub

   'Response Messages Ends

   'Creating the CRUD METHODS - STARTS

   '01. Get Method

   Private Function GetData() As DataTable



       command.CommandText = "SELECT * FROM tblProgramData"
```

```
    Dim adapter As New SQLiteDataAdapter(command)

    Dim dataTable As New DataTable()

    adapter.Fill(dataTable)

    Return dataTable

End Function

'Auto Data Refresh Method - STARTS

Private Sub RefreshDataGridView()

    frmProgram.dgvProgramData.DataSource = GetData()

    incrementedVariable()

End Sub

'Auto Data Refresh Method - ENDS

'Incremented Variable Starts

Public Sub incrementedVariable()

    ' Get the number of rows in the DataGridView

    Dim rowCount As Integer = frmProgram.dgvProgramData.Rows.Count

    ' Loop through each row and assign incremented values to the "Number" column

    For i As Integer = 0 To rowCount - 1

        frmProgram.dgvProgramData.Rows(i).Cells("No").Value = i + 1

    Next

End Sub

'Incremented Variable Ends
```

```
  '02. Update Method
  Private Sub UpdateData(ProgramID As Integer, ProgramLevel As String, ProgramName As String)
    command.CommandText = "UPDATE tblProgramData SET ProgramLevel = @ProgramLevel,  ProgramName = @ProgramName WHERE ProgramID = @ProgramID"
    command.Parameters.AddWithValue("@ProgramID", ProgramID)
    command.Parameters.AddWithValue("@ProgramLevel", ProgramLevel)
    command.Parameters.AddWithValue("@ProgramName", ProgramName)
    If (command.ExecuteNonQuery() > 0) Then
      ShowSuccessMessage()
    Else
      ShowFailureMessage()
    End If
  End Sub
  'Clear Data Method - STARTS
  Private Sub ClearData()
    cmbProgramLevel.SelectedItem = Nothing
    cmbProgram.SelectedItem = Nothing
  End Sub
  'Clear Data Method - ENDS
  'Handling User Interactions
```

```
    Private Sub btnUpdateData_Click(sender As Object, e As EventArgs) Handles
btnUpdateData.Click

        Dim ProgramID As Integer = txtID.Text

        Dim ProgramLevel As String = cmbProgramLevel.SelectedItem

        Dim ProgramName As String = cmbProgram.SelectedItem

        UpdateData(ProgramID, ProgramLevel, ProgramName)

        ClearData()

        RefreshDataGridView()

    End Sub

    Private Sub btnBack_Click(sender As Object, e As EventArgs) Handles btnBack.Click

        frmProgram.Show()

        Me.Close()

    End Sub

End Class
```

6. **Course Module**
    (a) View all courses

```
Imports System.Data.SQLite

Imports Microsoft.Office.Interop.Excel

Imports System.Runtime.InteropServices

Public Class frmCourse

    'Database Connections

    Private connection As SQLiteConnection

    Private command As SQLiteCommand
```

```
'Initializing SQLite Database connections
Public Sub New()
    InitializeComponent()
    Try
        connection = New SQLiteConnection("Data Source=gis.db;Version=3;Pooling=true;")
        connection.Open()
        command = connection.CreateCommand()
    Catch ex As Exception
        MessageBox.Show("Error connecting to the database: " & ex.Message, "Connection
Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
    End Try
End Sub
Private Sub frmCourse_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    CreateDatabaseTable()
    RefreshDataGridView()
    '...
    cmbSelectProgramLevel.SelectedIndex = 0
    cmbSelectProgram.SelectedIndex = 0
    cmbSelectYear.SelectedIndex = 0
    cmbSemester.SelectedIndex = 0
End Sub
```

```
    'Create database table if not exist

    Private Sub CreateDatabaseTable()

        command.CommandText = "CREATE TABLE IF NOT EXISTS tblCourseData (CourseID
INTEGER PRIMARY KEY AUTOINCREMENT, CourseName TEXT, CourseCode TEXT,
CreditHour TEXT, ProgramLevel TEXT, ProgramName TEXT, Year TEXT, Semester TEXT)"

        command.ExecuteNonQuery()

    End Sub

    'Response Messages Starts

    Private Sub ShowSuccessMessage()

        MessageBox.Show("Course Data successful removed!", "Success",
MessageBoxButtons.OK, MessageBoxIcon.Information)
    End Sub

    Private Sub ShowFailureMessage()

        MessageBox.Show("Failed to remove Course Data unsuccessful!", "Failure",
MessageBoxButtons.OK, MessageBoxIcon.Information)
    End Sub

    'Response Messages Ends

    'Creating the CRUD METHODS - STARTS

    '01. Get Method

    Private Function GetData() As Data.DataTable

        command.CommandText = "SELECT * FROM tblCourseData ORDER BY RANDOM()"

        Dim adapter As New SQLiteDataAdapter(command)

        Dim dataTable As New Data.DataTable()

        adapter.Fill(dataTable)
        Return DataTable

    End Function

    'Auto Data Refresh Method - STARTS

    Private Sub RefreshDataGridView()

        dgvCourseData.DataSource = GetData()

        incrementedVariable()

        dgvCourseData.Columns("No").DisplayIndex = 0

        dgvCourseData.Columns("CourseName").DisplayIndex = 1

        dgvCourseData.Columns("CourseCode").DisplayIndex = 2

        dgvCourseData.Columns("CreditHour").DisplayIndex = 3

        dgvCourseData.Columns("ProgramLevel").DisplayIndex = 4

        dgvCourseData.Columns("ProgramName").DisplayIndex = 5

        dgvCourseData.Columns("Year").DisplayIndex = 6

        dgvCourseData.Columns("Semester").DisplayIndex = 7

        dgvCourseData.Columns("Edit").DisplayIndex = 8

        dgvCourseData.Columns("Delete").DisplayIndex = 9

    End Sub

    'Auto Data Refresh Method - ENDS

    'Incremented Variable Starts

    Public Sub incrementedVariable()

        ' Get the number of rows in the DataGridView

        Dim rowCount As Integer = dgvCourseData.Rows.Count
```

```
        ' Loop through each row and assign incremented values to the "Number" column
        For i As Integer = 0 To rowCount - 1
            dgvCourseData.Rows(i).Cells("No").Value = i + 1
        Next
    End Sub
    'Incremented Variable Ends
    '02. Delete Method
    Private Sub DeleteData(CourseID As Integer)
        command.CommandText = "DELETE FROM tblCourseData WHERE CourseID = 
@CourseID"
        command.Parameters.AddWithValue("@CourseID", CourseID)
        If (command.ExecuteNonQuery() > 0) Then
            ShowSuccessMessage()
        Else
            ShowFailureMessage()
        End If
    End Sub
    'Creating the CRUD METHODS - ENDS
    'User Interaction
    '01. Create Click : Starts
    Private Sub btnCreate_Click(sender As Object, e As EventArgs) Handles btnCreate.Click
        frmCourseAddData.Show()
    End Sub
```

```
  '01. Create Click : Ends

  'Datagridview Cell Controls

  Private Sub dgvCourseData_CellContentClick(sender As Object, e As
DataGridViewCellEventArgs) Handles dgvCourseData.CellContentClick
      ' Check if the clicked cell is in the button column

      If e.ColumnIndex = 1 AndAlso e.RowIndex >= 0 Then

          ' Perform action when button is clicked

          Dim selectedRow As DataGridViewRow = dgvCourseData.Rows(e.RowIndex)

          'Get the data from the cells

          Dim column1Value As String = selectedRow.Cells("CourseID").Value.ToString()

          Dim column2Value As String = selectedRow.Cells("CourseName").Value.ToString()

          Dim column3Value As String = selectedRow.Cells("CourseCode").Value.ToString()

          Dim column4Value As String = selectedRow.Cells("CreditHour").Value.ToString()

          Dim column5Value As String = selectedRow.Cells("ProgramLevel").Value.ToString()

          Dim column6Value As String = selectedRow.Cells("ProgramName").Value.ToString()

          Dim column7Value As String = selectedRow.Cells("Year").Value.ToString()

          Dim column8Value As String = selectedRow.Cells("Semester").Value.ToString()

          ' Create an instance of the target form

          Dim targetForm As New frmCourseUpdateData()

          ' Pass the data to the target form

          targetForm.SetData(column1Value, column2Value, column3Value, column4Value,
column5Value, column6Value, column7Value, column8Value)
```

```
            ' Show the target form

            targetForm.Show()

        ElseIf e.ColumnIndex = 2 AndAlso e.RowIndex >= 0 Then

            ' Display a question prompt and get user's choice

            Dim userChoice As DialogResult = MessageBox.Show("Do you want to delete this
particular Course?", "Delete", MessageBoxButtons.YesNo, MessageBoxIcon.Question)

            ' Check the user's choice and perform actions accordingly

            If userChoice = DialogResult.Yes Then

                ' Code to execute when user chooses Yes

                Dim selectedRow As DataGridViewRow = dgvCourseData.Rows(e.RowIndex)

                Dim dataFromRow As Integer = selectedRow.Cells("CourseID").Value

                Dim id As Integer = dataFromRow

                ' ...

                DeleteData(id)

                RefreshDataGridView()

            End If

        End If

    End Sub

    'Filter Button Logic Starts

    Public Sub ShowFilterResult(ProgramLevel As String, ProgramName As String, Year As
Integer, Semester As String)
```

```
        Dim searchQuery As String = "SELECT * FROM tblCourseData WHERE ProgramLevel LIKE '%' || @ProgramLevel || '%' AND ProgramName LIKE '%' || @ProgramName || '%' AND Year LIKE '%' || @Year || '%'  AND Semester LIKE '%' || @Semester || '%'"
        Dim filterCmd As New SQLiteCommand(searchQuery, connection)
        ' Use parameterized query to avoid SQL injection
        filterCmd.Parameters.AddWithValue("@ProgramLevel", ProgramLevel)
        filterCmd.Parameters.AddWithValue("@ProgramName", ProgramName)
        filterCmd.Parameters.AddWithValue("@Year", Year)
        filterCmd.Parameters.AddWithValue("@Semester", Semester)
        Dim resultAdapter As New SQLiteDataAdapter
        resultAdapter.SelectCommand = filterCmd
        ' Create an instance of the DataTable class
        Dim resultDataTable As New Data.DataTable()
        resultAdapter.Fill(resultDataTable)
        dgvCourseData.DataSource = resultDataTable
        incrementedVariable()
    End Sub
    Private Sub btnFilterSearchB_Click(sender As Object, e As EventArgs) Handles btnFilterSearchB.Click
        Dim ProgramLevel As String = cmbSelectProgramLevel.SelectedItem
        Dim ProgramName As String = cmbSelectProgram.SelectedItem
        Dim Year As String = cmbSelectYear.SelectedItem
        Dim Semester As String = cmbSemester.SelectedItem
        ShowFilterResult(ProgramLevel, ProgramName, Year, Semester)

    End Sub

    Private Sub btnResetB_Click(sender As Object, e As EventArgs) Handles btnResetB.Click

        RefreshDataGridView()

        '...

        cmbSelectProgramLevel.SelectedIndex = 0

        cmbSelectProgram.SelectedIndex = 0

        cmbSelectYear.SelectedIndex = 0

        cmbSemester.SelectedIndex = 0

    End Sub

    'Filter Button Logic Ends

    'Form Controls

    Private Sub btnBack_Click_1(sender As Object, e As EventArgs) Handles btnBack.Click

        frmDashboard.Show()

        Me.Close()

    End Sub

End Class
```

(b) Add New Course

```
Imports System.Data.SQLite

Public Class frmCourseAddData

    'Database Connections

    Private connection As SQLiteConnection
    Private command As SQLiteCommand
    'Initializing SQLite Database connections
    Public Sub New()
        InitializeComponent()
        Try
            connection = New SQLiteConnection("Data Source=gis.db;Version=3;Pooling=true;")
            connection.Open()
            command = connection.CreateCommand()
        Catch ex As Exception
            MessageBox.Show("Error connecting to the database: " & ex.Message, "Connection
Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
        End Try
    End Sub
    Private Sub frmCourseAddData_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        cmbCreditHour.SelectedIndex = 0
        cmbSemester.SelectedIndex = 0
        cmbProgramLevel.SelectedIndex = 0
        cmbProgram.SelectedIndex = 0
        cmbYear.SelectedIndex = 0
    End Sub
    'Response Messages Starts
    Private Sub ShowSuccessMessage()
```

```
        MessageBox.Show("Course created successfully!", "Success", MessageBoxButtons.OK,
MessageBoxIcon.Information)

    End Sub

    Private Sub ShowFailureMessage()

        MessageBox.Show("Course creation unsuccessful!", "Failure", MessageBoxButtons.OK,
MessageBoxIcon.Information)

    End Sub

    'Response Messages Ends

    'Creating the CRUD METHODS - STARTS

    '01. Insert Method

    Private Sub InsertData(ByVal coursename As String, ByVal coursecode As String, ByVal
credithour As String, ByVal programLevel As String, ByVal programName As String, ByVal
year As String, ByVal semester As String)

        Dim insertQuery As String = "INSERT INTO tblCourseData (CourseName, CourseCode,
CreditHour, ProgramLevel, ProgramName, Year, Semester) VALUES (@CourseName,
@CourseCode, @CreditHour, @ProgramLevel, @ProgramName, @Year, @Semester);"

        Using cmd As New SQLiteCommand(insertQuery, connection)

            cmd.Parameters.AddWithValue("@CourseName", coursename)

            cmd.Parameters.AddWithValue("@CourseCode", coursecode)

            cmd.Parameters.AddWithValue("@CreditHour", credithour)

            cmd.Parameters.AddWithValue("@ProgramLevel", programLevel)

            cmd.Parameters.AddWithValue("@ProgramName", programName)

            cmd.Parameters.AddWithValue("@Year", year)

            cmd.Parameters.AddWithValue("@Semester", semester)
            If (cmd.ExecuteNonQuery() > 0) Then
        ShowSuccessMessage()
            Else
                ShowFailureMessage()
            End If
        End Using
    End Sub
    '02. Get Data Method
    Private Function GetData() As DataTable
        command.CommandText = "SELECT * FROM tblCourseData"
        Dim adapter As New SQLiteDataAdapter(command)
        Dim dataTable As New DataTable()
        adapter.Fill(dataTable)
        Return dataTable
    End Function
    'Creating the CRUD METHODS - ENDS
    'Auto Data Refresh Method - STARTS
    Private Sub RefreshDataGridView()
        frmCourse.dgvCourseData.DataSource = GetData()
        incrementedVariable()
    End Sub
```

```
'Auto Data Refresh Method - ENDS

'Incremented Variable Starts

Public Sub incrementedVariable()

    ' Get the number of rows in the DataGridView

    Dim rowCount As Integer = frmCourse.dgvCourseData.Rows.Count

    ' Loop through each row and assign incremented values to the "Number" column

    For i As Integer = 0 To rowCount - 1

        frmCourse.dgvCourseData.Rows(i).Cells("No").Value = i + 1

    Next

End Sub

'Incremented Variable Ends

'Clear Data Method - STARTS

Private Sub ClearData()

    txtCourseName.Text = " "

    txtCourseCode.Text = " "

    cmbCreditHour.SelectedItem = Nothing

    cmbSemester.SelectedItem = Nothing

    cmbProgramLevel.SelectedItem = Nothing

    cmbProgram.SelectedItem = Nothing

    cmbYear.SelectedItem = Nothing

End Sub
```

```
'Clear Data Method - ENDS

'Handling User Interactions

Private Sub btnInsertData_Click(sender As Object, e As EventArgs) Handles
btnInsertData.Click

    Dim coursename As String = txtCourseName.Text

    Dim coursecode As String = txtCourseCode.Text

    Dim credithour As String = cmbCreditHour.SelectedItem

    Dim Programlevel As String = cmbProgramLevel.SelectedItem

    Dim ProgramName As String = cmbProgram.SelectedItem

    Dim Year As String = cmbYear.SelectedItem

    Dim semester As String = cmbSemester.SelectedItem

    InsertData(coursename, coursecode, credithour, Programlevel, ProgramName, Year,
semester)

    ClearData()

    RefreshDataGridView()

End Sub

'Form Controls

Private Sub btnBack_Click(sender As Object, e As EventArgs) Handles btnBack.Click

    frmCourse.Show()

    Me.Hide()

End Sub
End Class
```

(c) Edit Course

```
Imports System.Data.SQLite

Public Class frmCourseUpdateData

    'Database Connections

    Private connection As SQLiteConnection

    Private command As SQLiteCommand

    'Initializing SQLite Database connections

    Public Sub New()

        InitializeComponent()

        Try

            connection = New SQLiteConnection("Data Source=gis.db;Version=3;Pooling=true;")

            connection.Open()

            command = connection.CreateCommand()

        Catch ex As Exception

            MessageBox.Show("Error connecting to the database: " & ex.Message, "Connection
Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

        End Try

    End Sub

    ' Properties to store the data

    Public Property CourseID As Integer

    Public Property CourseName As String

    Public Property CourseCode As String
    Public Property CreditHour As String

    Public Property ProgramLevel As String

    Public Property ProgramName As String

    Public Property Year As String

    Public Property Semester As String

    ' Method to set the data

    Public Sub SetData(column1Value As String, column2Value As String, column3Value As String, column4Value As String, column5Value As String, column6Value As String, column7Value As String, column8Value As String)

        CourseID = Integer.Parse(column1Value)

        CourseName = column2Value

        CourseCode = column3Value

        CreditHour = column4Value

        ProgramLevel = column5Value

        ProgramName = column6Value

        Year = column7Value

        Semester = column8Value

    End Sub

    Private Sub frmCourseUpdateData_Load(sender As Object, e As EventArgs) Handles MyBase.Load

        ' Display the data in the labels, textboxes, etc. on the TargetForm

        txtID.Text = CourseID

        txtCourseName.Text = CourseName
        txtCourseCode.Text = CourseCode

        cmbCreditHour.SelectedItem = CreditHour

        cmbProgramLevel.SelectedItem = ProgramLevel

        cmbProgram.SelectedItem = ProgramName

        cmbYear.SelectedItem = Year

        cmbSemester.SelectedItem = Semester

        ' To hide the textbox for Unique ID

        txtID.Visible = False

        ' ...

    End Sub

    'Response Messages Starts

    Private Sub ShowSuccessMessage()
        MessageBox.Show("Course Data updated successful!", "Success",
MessageBoxButtons.OK, MessageBoxIcon.Information)
    End Sub

    Private Sub ShowFailureMessage()

       MessageBox.Show("Course Data update unsuccessful!", "Failure", MessageBoxButtons.OK,
MessageBoxIcon.Information)
    End Sub

    'Response Messages Ends

    'Creating the CRUD METHODS - STARTS

    '01. Get Method

    Private Function GetData() As DataTable

        command.CommandText = "SELECT * FROM tblCourseData"

        Dim adapter As New SQLiteDataAdapter(command)
        Dim dataTable As New DataTable()

        adapter.Fill(dataTable)

        Return dataTable

    End Function

    'Auto Data Refresh Method - STARTS

    Private Sub RefreshDataGridView()

        frmCourse.dgvCourseData.DataSource = GetData()

        incrementedVariable()

    End Sub

    'Auto Data Refresh Method - ENDS

    'Incremented Variable Starts

    Public Sub incrementedVariable()

        ' Get the number of rows in the DataGridView

        Dim rowCount As Integer = frmCourse.dgvCourseData.Rows.Count

        ' Loop through each row and assign incremented values to the "Number" column

        For i As Integer = 0 To rowCount - 1

            frmCourse.dgvCourseData.Rows(i).Cells("No").Value = i + 1

        Next

    End Sub

    'Incremented Variable Ends
```

```
'02. Update Method

Private Sub UpdateData(CourseID As Integer, CourseName As String, CourseCode As String, CreditHour As String, ProgramLevel As String, ProgramName As String, Year As String, Semester As String)

    command.CommandText = "UPDATE tblCourseData SET CourseName = @CourseName, CourseCode = @CourseCode, CreditHour = @CreditHour, ProgramLevel = @ProgramLevel, ProgramName = @ProgramName, Year = @Year, Semester = @Semester WHERE CourseID = @CourseID"

    command.Parameters.AddWithValue("@CourseID", CourseID)

    command.Parameters.AddWithValue("@CourseName", CourseName)

    command.Parameters.AddWithValue("@CourseCode", CourseCode)

    command.Parameters.AddWithValue("@CreditHour", CreditHour)

    command.Parameters.AddWithValue("@ProgramLevel", ProgramLevel)

    command.Parameters.AddWithValue("@ProgramName", ProgramName)

    command.Parameters.AddWithValue("@Year", Year)

    command.Parameters.AddWithValue("@Semester", Semester)

    If (command.ExecuteNonQuery() > 0) Then

      ShowSuccessMessage()

    Else

      ShowFailureMessage()

    End If

End Sub
```

```
    'Clear Data Method - STARTS
    Private Sub ClearData()
        txtCourseName.Text = ""
        txtCourseCode.Text = ""
        cmbCreditHour.SelectedIndex = Nothing
        cmbProgramLevel.SelectedItem = Nothing
        cmbProgram.SelectedItem = Nothing
        cmbYear.SelectedItem = Nothing
        cmbSemester.SelectedIndex = Nothing
    End Sub
    'Clear Data Method - ENDS
    'Handling User Interactions
    Private Sub btnUpdateData_Click(sender As Object, e As EventArgs) Handles
btnUpdateData.Click
        Dim CourseID As Integer = txtID.Text
        Dim CourseName As String = txtCourseName.Text
        Dim CourseCode As String = txtCourseCode.Text
        Dim MreditHour As String = cmbCreditHour.SelectedItem
        Dim ProgramLevel As String = cmbProgramLevel.SelectedItem
        Dim ProgramName As String = cmbProgram.SelectedItem
        Dim Year As String = cmbYear.SelectedItem
        Dim Semester As String = cmbSemester.SelectedItem
        UpdateData(CourseID, CourseName, CourseCode, CreditHour, ProgramLevel, ProgramName, Year, Semester)
        ClearData()
        RefreshDataGridView()
    End Sub
    'Form Controls
    Private Sub btnBack_Click(sender As Object, e As EventArgs) Handles btnBack.Click
        frmCourse.Show()
        Me.Close()
    End Sub
End Class
```

7. Grade book Module

```
Imports System.Data.SQLite
Public Class frmGrade
    'Database Connections
    Private connection As SQLiteConnection
    Private command As SQLiteCommand
    'Initializing SQLite Database connections
    Public Sub New()
        InitializeComponent()
        Try
            connection = New SQLiteConnection("Data Source=gis.db;Version=3;Pooling=true;")
            connection.Open()
            command = connection.CreateCommand()
        Catch ex As Exception
            MessageBox.Show("Error connecting to the database: " & ex.Message, "Connection
    Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
        End Try
    End Sub
    Private Sub frmGrade_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        CreateDatabaseTable()
        RefreshDataGridView()
        cmbSelectProgramLevel.SelectedIndex = 0
        cmbSelectProgram.SelectedIndex = 0
        cmbSelectYear.SelectedIndex = 0
    End Sub
    'Create database table If Not exist
    Private Sub CreateDatabaseTable()
        command.CommandText = "CREATE TABLE IF NOT EXISTS tblGradeData
        (GradeID INTEGER PRIMARY KEY AUTOINCREMENT,
        StudID INTEGER,
        CourseID INTEGER,
        Grade TEXT,
        Semester TEXT,
        FOREIGN KEY (StudID) REFERENCES tblStudentData(StudID),
        FOREIGN KEY (CourseID) REFERENCES tblCourseData(CourseID))"

        command.ExecuteNonQuery()

    End Sub

    'Response Messages Starts

    Private Sub ShowSuccessMessage()

        MessageBox.Show("Grade Data successful removed!", "Success",
MessageBoxButtons.OK, MessageBoxIcon.Information)
    End Sub

    Private Sub ShowFailureMessage()

       MessageBox.Show("Grade Data removal unsuccessful!", "Failure",
MessageBoxButtons.OK, MessageBoxIcon.Information)
    End Sub

    'Response Messages Ends

    'Creating the CRUD METHODS - STARTS

    '01. Get Method

    Private Function GetData() As Data.DataTable

        command.CommandText = "SELECT StudID, FirstName, LastName, StudentID, Year,
ProgramLevel, ProgramName FROM tblStudentData ORDER BY RANDOM() LIMIT 100"

        Dim adapter As New SQLiteDataAdapter(command)

        Dim dataTable As New Data.DataTable()

        adapter.Fill(dataTable)

        Return dataTable

    End Function
```

```
'Auto Data Refresh Method - STARTS

Private Sub RefreshDataGridView()

    dgvStudentGradeData.DataSource = GetData()

    incrementedVariable()

    dgvStudentGradeData.Columns("No").DisplayIndex = 0

    dgvStudentGradeData.Columns("FirstName").DisplayIndex = 1

    dgvStudentGradeData.Columns("LastName").DisplayIndex = 2

    dgvStudentGradeData.Columns("StudentID").DisplayIndex = 3

    dgvStudentGradeData.Columns("Year").DisplayIndex = 4

    dgvStudentGradeData.Columns("btnAdd").DisplayIndex = 5

End Sub

'Auto Data Refresh Method - ENDS

'Incremented Variable Starts

Public Sub incrementedVariable()

    ' Get the number of rows in the DataGridView

    Dim rowCount As Integer = dgvStudentGradeData.Rows.Count

    ' Loop through each row and assign incremented values to the "Number" column

    For i As Integer = 0 To rowCount - 1

        dgvStudentGradeData.Rows(i).Cells("No").Value = i + 1

    Next

End Sub

'Incremented Variable Ends
```

```
  'User Interaction

  '01. Create Click : Starts with Datagrid Cellcontent click event

  Private Sub dgvStudentGradeData_CellContentClick(sender As Object, e As
DataGridViewCellEventArgs) Handles dgvStudentGradeData.CellContentClick
      ' Check if the clicked cell is in the button column

      If e.ColumnIndex = 1 AndAlso e.RowIndex >= 0 Then

          ' Perform action when button is clicked

          Dim selectedRow As DataGridViewRow = dgvStudentGradeData.Rows(e.RowIndex)

          'Get the data from the cells

          Dim column1Value As String = selectedRow.Cells("StudID").Value.ToString()

          Dim column2Value As String = selectedRow.Cells("FirstName").Value.ToString()

          Dim column3Value As String = selectedRow.Cells("LastName").Value.ToString()

          Dim column4Value As String = selectedRow.Cells("StudentID").Value.ToString()

          Dim column5Value As String = selectedRow.Cells("Year").Value.ToString()

          Dim column6Value As String = selectedRow.Cells("ProgramLevel").Value.ToString()

          Dim column7Value As String = selectedRow.Cells("ProgramName").Value.ToString()

          ' Create an instance of the target form

          Dim targetForm As New frmGradeAddData()

          ' Pass the data to the target form

          targetForm.SetData(column1Value, column2Value, column3Value, column4Value,
column5Value, column6Value, column7Value)

          ' Show the target form

          targetForm.Show()
```

```
        End If
    End Sub
    '01. Create Click : Ends with Datagrid Cellcontent click event
    'Search Button Logic Starts
    Public Sub ShowFilterResult(ProgramLevel As String, ProgramName As String, Year As
Integer)
        Dim searchQuery As String = "SELECT StudID, FirstName, LastName, StudentID, Year,
ProgramLevel, ProgramName FROM tblStudentData WHERE ProgramLevel LIKE '%' ||
@ProgramLevel || '%' AND ProgramName LIKE '%' || @ProgramName || '%' AND Year LIKE
'%' || @Year || '%'"
        Dim filterCmd As New SQLiteCommand(searchQuery, connection)
        ' Use parameterized query to avoid SQL injection
        filterCmd.Parameters.AddWithValue("@ProgramLevel", ProgramLevel)
        filterCmd.Parameters.AddWithValue("@ProgramName", ProgramName)
        filterCmd.Parameters.AddWithValue("@Year", Year)
        Dim resultAdapter As New SQLiteDataAdapter
        resultAdapter.SelectCommand = filterCmd
        ' Create an instance of the DataTable class
        Dim resultDataTable As New Data.DataTable()
        resultAdapter.Fill(resultDataTable)
        dgvStudentGradeData.DataSource = resultDataTable
        incrementedVariable()
        dgvStudentGradeData.Columns("No").DisplayIndex = 0
        dgvStudentGradeData.Columns("FirstName").DisplayIndex = 1
        dgvStudentGradeData.Columns("LastName").DisplayIndex = 2
        dgvStudentGradeData.Columns("StudentID").DisplayIndex = 3
        dgvStudentGradeData.Columns("Year").DisplayIndex = 4
        dgvStudentGradeData.Columns("btnAdd").DisplayIndex = 5
    End Sub
    Private Sub btnFilterSearchC_Click(sender As Object, e As EventArgs) Handles
btnFilterSearchC.Click
        Dim ProgramLevel As String = cmbSelectProgramLevel.SelectedItem
        Dim ProgramName As String = cmbSelectProgram.SelectedItem
        Dim Year As String = cmbSelectYear.SelectedItem
        ShowFilterResult(ProgramLevel, ProgramName, Year)
    End Sub
    Private Sub btnReset_Click(sender As Object, e As EventArgs) Handles btnReset.Click
        RefreshDataGridView()
        '...
        cmbSelectProgramLevel.SelectedIndex = 0
        cmbSelectProgram.SelectedIndex = 0
        cmbSelectYear.SelectedIndex = 0
    End Sub
    'Search Button Logic Ends
    'Form Control
    Private Sub btnBack_Click(sender As Object, e As EventArgs) Handles btnBack.Click
        frmDashboard.Show()
        Me.Close()
    End Sub
End Class
```

(a) Add Student Grade Logic

```
Imports System.Data.SQLite
Public Class frmGradeAddData
    'Database Connections
    Private connection As SQLiteConnection
    Private command As SQLiteCommand
    'Initializing SQLite Database connections
    Public Sub New()
        InitializeComponent()
        Try
            connection = New SQLiteConnection("Data Source=gis.db;Version=3;Pooling=true;")
            connection.Open()
            command = connection.CreateCommand()
        Catch ex As Exception
            MessageBox.Show("Error connecting to the database: " & ex.Message, "Connection
Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
        End Try
    End Sub
    ' Properties to store the data
    Public Property StudID As Integer
    Public Property FirstName As String

    Public Property LastName As String

    Public Property StudentID As Integer

    Public Property Year As Integer

    Public Property ProgramLevel As String

    Public Property ProgramName As String

    Dim selectedValue As Integer

    ' Method to set the data

    Public Sub SetData(column1Value As String, column2Value As String, column3Value As
String, column4Value As String, column5Value As String, column6Value As String,
column7Value As String)

        StudID = Integer.Parse(column1Value)

        FirstName = column2Value

        LastName = column3Value

        StudentID = Integer.Parse(column4Value)

        Year = Integer.Parse(column5Value)

        ProgramLevel = column6Value

        ProgramName = column7Value

    End Sub

    Private Sub frmGradeAddData_Load(sender As Object, e As EventArgs) Handles
MyBase.Load

        ' Display the data in the labels, textboxes, etc. on the TargetForm

        txtStudID.Text = StudID
        txtFName.Text = FirstName

        txtLName.Text = LastName

        txtStudentID.Text = StudentID

        ' To hide the textbox for Unique ID

        txtStudID.Visible = False

        txtYear.Visible = False

        ' ...

        cmbSelectLetterGrade.SelectedIndex = 0

        cmbSemester.SelectedIndex = 0

        GetCourseData(Year, ProgramLevel, ProgramName)

    End Sub

    'Response Messages Starts

    Private Sub ShowSuccessMessage()

       MessageBox.Show("Student Grade successful entered!", "Success",
MessageBoxButtons.OK, MessageBoxIcon.Information)
    End Sub

    Private Sub ShowFailureMessage()

        MessageBox.Show("Fail to enter student grade!", "Failure", MessageBoxButtons.OK,
MessageBoxIcon.Information)

    End Sub

    'Response Messages Ends

    'Creating the CRUD METHODS - STARTS

    '01. Get Method
```

```
  Public Sub GetCourseData(Year As Integer, ProgramLevel As String,  ProgramName As String)

    Dim query As String = "SELECT * FROM tblCourseData WHERE Year = @Year AND ProgramLevel = @ProgramLevel AND ProgramName = @ProgramName"

    Dim command As New SQLiteCommand(query, connection)

    ' Use parameterized query to avoid SQL injection

    command.Parameters.AddWithValue("@Year", Year)

    command.Parameters.AddWithValue("@ProgramLevel", ProgramLevel)

    command.Parameters.AddWithValue("@ProgramName", ProgramName)

    Dim dataAdapter As New SQLiteDataAdapter

    dataAdapter.SelectCommand = command

    Dim DT As New DataTable()

    ' Fill DataTable with data

    dataAdapter.Fill(DT)

    ' Insert default row

    Dim row As DataRow = DT.NewRow

    row("CourseID") = 0

    row("CourseName") = "Select Course"

    DT.Rows.InsertAt(row, 0)

    cmbCourse.DataSource = DT

    cmbCourse.DisplayMember = "CourseName"

    cmbCourse.ValueMember = "CourseID"

  End Sub
```

```
    Private Function GetData() As Data.DataTable
        command.CommandText = "SELECT StudID, FirstName, LastName, StudentID, Year,
ProgramLevel, ProgramName FROM tblStudentData ORDER BY RANDOM() LIMIT 100"
        Dim adapter As New SQLiteDataAdapter(command)
        Dim dataTable As New Data.DataTable()
        adapter.Fill(dataTable)
        Return dataTable
    End Function
    'Auto Data Refresh Method - STARTS
    Private Sub RefreshDataGridView()
        frmGrade.dgvStudentGradeData.DataSource = GetData()
        incrementedVariable()
        frmGrade.dgvStudentGradeData.Columns("No").DisplayIndex = 0
        frmGrade.dgvStudentGradeData.Columns("FirstName").DisplayIndex = 1
        frmGrade.dgvStudentGradeData.Columns("LastName").DisplayIndex = 2
        frmGrade.dgvStudentGradeData.Columns("StudentID").DisplayIndex = 3
        frmGrade.dgvStudentGradeData.Columns("Year").DisplayIndex = 4
        frmGrade.dgvStudentGradeData.Columns("btnAdd").DisplayIndex = 5
    End Sub
    'Auto Data Refresh Method - ENDS
    'Incremented Variable Starts
    Public Sub incrementedVariable()
```

```
        ' Get the number of rows in the DataGridView
        Dim rowCount As Integer = frmGrade.dgvStudentGradeData.Rows.Count
        ' Loop through each row and assign incremented values to the "Number" column
        For i As Integer = 0 To rowCount - 1
            frmGrade.dgvStudentGradeData.Rows(i).Cells("No").Value = i + 1
        Next
    End Sub
    'Incremented Variable Ends
    '02. Update Method
    Private Sub InsertData(StudID As Integer, CourseID As Integer, Grade As String, Semester As
String)
        Dim insertQuery As String = "INSERT INTO tblGradeData (StudID, CourseID, Grade,
Semester) VALUES (@StudID, @CourseID, @Grade, @Semester);"
        Using cmd As New SQLiteCommand(insertQuery, connection)
            cmd.Parameters.AddWithValue("@StudID", StudID)
            cmd.Parameters.AddWithValue("@CourseID", CourseID)
            cmd.Parameters.AddWithValue("@Grade", Grade)
            cmd.Parameters.AddWithValue("@Semester", Semester)
            If (cmd.ExecuteNonQuery() > 0) Then
                ShowSuccessMessage()
            Else
                ShowFailureMessage()
            End If

        End Using

    End Sub

    'Clear Data Method - STARTS

    Private Sub ClearData()

        txtFName.Text = " "

        txtLName.Text = " "

        txtStudentID.Text = " "

        cmbCourse.SelectedItem = Nothing

        cmbSelectLetterGrade.SelectedItem = Nothing

        cmbSemester.SelectedItem = Nothing

    End Sub

    'Clear Data Method - ENDS

    Private Sub ChangeRowBGColor()

        ' Check if a row is selected.

        If frmGrade.dgvStudentGradeData.SelectedRows.Count > 0 Then

            ' Get the selected row.

          Dim selectedRow As DataGridViewRow =
frmGrade.dgvStudentGradeData.SelectedRows(0)
            ' Change the background color of the selected row.

            selectedRow.DefaultCellStyle.BackColor = Color.Yellow ' Change to your desired color.

        Else

            MessageBox.Show("Please select a row first.")

        End If
    End Sub

    'Handling User Interactions

    Private Sub btnInsertData_Click(sender As Object, e As EventArgs) Handles btnInsertData.Click

        Dim StudID As Integer = txtStudID.Text

        Dim CourseID As Integer = cmbCourse.SelectedValue

        Dim Grade As String = cmbSelectLetterGrade.SelectedItem

        Dim Semester As String = cmbSemester.SelectedItem

        InsertData(StudID, CourseID, Grade, Semester)

        RefreshDataGridView()

        ClearData()

        ChangeRowBGColor()

    End Sub

    'Form Controls

    Private Sub btnBack_Click(sender As Object, e As EventArgs) Handles btnBack.Click

        frmGrade.Show()

        Me.Close()

    End Sub

End Class
```

(b) Modify Student Grade Logic (Edit/Delete Grade)

1. **General View of all Grades**

```
Imports System.Data.SQLite

Imports System.IO
Imports iTextSharp.text

Imports iTextSharp.text.pdf

Imports Microsoft.Office.Interop.Excel

Imports System.Runtime.InteropServices

Public Class frmGradeModify

    'Database Connections

    Private connection As SQLiteConnection

    Private command As SQLiteCommand

    'Initializing SQLite Database connections

    Public Sub New()

        InitializeComponent()

        Try

            connection = New SQLiteConnection("Data Source=gis.db;Version=3;Pooling=true;")

            connection.Open()

            command = connection.CreateCommand()

        Catch ex As Exception

            MessageBox.Show("Error connecting to the database: " & ex.Message, "Connection
Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

        End Try

    End Sub

    Private Sub frmViewGrade_Load(sender As Object, e As EventArgs) Handles MyBase.Load

        RefreshDataGridView()

        '...
        cmbSelectProgramLevel.SelectedIndex = 0

        cmbSelectProgram.SelectedIndex = 0

        cmbSelectYear.SelectedIndex = 0

        cmbSemester.SelectedIndex = 0

    End Sub

    'Response Messages Starts

    Private Sub ShowSuccessMessage()

       MessageBox.Show("Grade Data successful removed!", "Success", MessageBoxButtons.OK,
MessageBoxIcon.Information)
    End Sub

    Private Sub ShowFailureMessage()

       MessageBox.Show("Failed to remove Grade Data unsuccessful!", "Failure",
MessageBoxButtons.OK, MessageBoxIcon.Information)
    End Sub

    Private Sub ShowSuccessMessageImport()

       MessageBox.Show("Course Data successful Imported!", "Success",
MessageBoxButtons.OK, MessageBoxIcon.Information)
    End Sub

    Private Sub ShowFailureMessageImport()

        MessageBox.Show("Failed to Import Course Data!", "Failure", MessageBoxButtons.OK,
MessageBoxIcon.Information)

    End Sub

    'Response Messages Ends

    'Creating the CRUD METHODS - STARTS

    '01. Get Method

    Private Function GetData() As Data.DataTable
```

```
        command.CommandText = "SELECT DISTINCT GradeID, Grade, tblGradeData.Semester,
    tblCourseData.CourseName, tblCourseData.Year, tblStudentData.FirstName,
    tblStudentData.LastName,    tblStudentData.StudentID,    tblStudentData.ProgramLevel    As
ProgramLevel, tblStudentData.ProgramName As ProgramName
    FROM tblGradeData
    JOIN tblCourseData ON tblGradeData.CourseID = tblCourseData.CourseID
    JOIN tblStudentData ON tblGradeData.StudID = tblStudentData.StudID
    ORDER BY RANDOM()"
        Dim adapter As New SQLiteDataAdapter(command)
        Dim dataTable As New Data.DataTable()
        adapter.Fill(dataTable)
        Return dataTable
    End Function
    'Auto Data Refresh Method - STARTS
    Private Sub RefreshDataGridView()
        dgvViewStudentGradeData.DataSource = GetData()
        incrementedVariable()
        dgvViewStudentGradeData.Columns("No").DisplayIndex = 0
        dgvViewStudentGradeData.Columns("FirstName").DisplayIndex = 1
        dgvViewStudentGradeData.Columns("LastName").DisplayIndex = 2
        dgvViewStudentGradeData.Columns("StudentID").DisplayIndex = 3
        dgvViewStudentGradeData.Columns("CourseName").DisplayIndex = 4
        dgvViewStudentGradeData.Columns("Grade").DisplayIndex = 5
        dgvViewStudentGradeData.Columns("Semester").DisplayIndex = 6

        dgvViewStudentGradeData.Columns("Edit").DisplayIndex = 7

        dgvViewStudentGradeData.Columns("Delete").DisplayIndex = 8

    End Sub

    'Auto Data Refresh Method - ENDS

    'Incremented Variable Starts

    Public Sub incrementedVariable()

        ' Get the number of rows in the DataGridView

        Dim rowCount As Integer = dgvViewStudentGradeData.Rows.Count

        ' Loop through each row and assign incremented values to the "Number" column

        For i As Integer = 0 To rowCount - 1

            dgvViewStudentGradeData.Rows(i).Cells("No").Value = i + 1

        Next

    End Sub

    'Incremented Variable Ends

    '02. Delete Method

    Private Sub DeleteData(GradeID As Integer)

        command.CommandText = "DELETE FROM tblGradeData WHERE GradeID =
@GradeID"

        command.Parameters.AddWithValue("@GradeID", GradeID)

        If (command.ExecuteNonQuery() > 0) Then

            ShowSuccessMessage()

        Else
            ShowFailureMessage()

        End If

    End Sub

    'Creating the CRUD METHODS - ENDS

    'Handling User Interaction

    '01. Create Click : Starts with Datagrid Cellcontent click event

    Private Sub dgvViewStudentGradeData_CellContentClick(sender As Object, e As
DataGridViewCellEventArgs) Handles dgvViewStudentGradeData.CellContentClick

        ' Check if the clicked cell is in the button column

        If e.ColumnIndex = 2 AndAlso e.RowIndex >= 0 Then

            ' Perform action when button is clicked

            Dim selectedRow As DataGridViewRow =
dgvViewStudentGradeData.Rows(e.RowIndex)
            'Get the data from the cells

            Dim column1Value As String = selectedRow.Cells("GradeID").Value.ToString()

            Dim column2Value As String = selectedRow.Cells("FirstName").Value.ToString()

            Dim column3Value As String = selectedRow.Cells("LastName").Value.ToString()

            Dim column4Value As String = selectedRow.Cells("StudentID").Value.ToString()

            Dim column5Value As String = selectedRow.Cells("CourseName").Value.ToString()

            Dim column6Value As String = selectedRow.Cells("Grade").Value.ToString()

            Dim column7Value As String = selectedRow.Cells("Year").Value.ToString()

            Dim column8Value As String = selectedRow.Cells("Semester").Value.ToString()

            Dim column9Value As String = selectedRow.Cells("ProgramLevel").Value.ToString()

            Dim column10Value As String = selectedRow.Cells("ProgramName").Value.ToString()
```

```
            ' Create an instance of the target form

            Dim targetForm As New frmGradeUpdateData()

            ' Pass the data to the target form

            targetForm.SetData(column1Value, column2Value, column3Value, column4Value,
column5Value, column6Value, column7Value, column8Value, column9Value, column10Value)

            ' Show the target form

            targetForm.Show()

        ElseIf e.ColumnIndex = 3 AndAlso e.RowIndex >= 0 Then

            ' Perform action when button is clicked

            ' Display a question prompt and get user's choice

            Dim userChoice As DialogResult = MessageBox.Show("Do you want To delete this
particular Grade?", "Delete", MessageBoxButtons.YesNo, MessageBoxIcon.Question)

            ' Check the user's choice and perform actions accordingly

            If userChoice = DialogResult.Yes Then

                ' Code to execute when user chooses Yes

                Dim selectedRow As DataGridViewRow =
dgvViewStudentGradeData.Rows(e.RowIndex)
                Dim dataFromRow As Integer = selectedRow.Cells("GradeID").Value

                Dim id As Integer = dataFromRow

                ' ...

                DeleteData(id)

                RefreshDataGridView()

            End If

        End If
End Sub

'01. Create Click : Ends with Datagrid Cellcontent click event

'02. Excel Export Click : Starts

Private Sub btnExportGradeExcel_Click(sender As Object, e As EventArgs) Handles
btnExportGradeExcel.Click

        ' Display a question prompt and get user's choice

        Dim userChoice As DialogResult = MessageBox.Show("Do you want To Export Student
Grades?", "Question", MessageBoxButtons.YesNo, MessageBoxIcon.Question)

        ' Check the user's choice and perform actions accordingly

        If userChoice = DialogResult.Yes Then

            ' Code to execute when user chooses Yes

            ' Initialize Excel application

            Dim excelApp As New Application()

            Dim workbook As Workbook = excelApp.Workbooks.Add()

            Dim worksheet As Worksheet = workbook.Sheets(1)

            Dim sqlQuery As String = "Select Grade, tblCourseData.CourseName,
tblStudentData.FirstName, tblStudentData.LastName, tblStudentData.StudentID

                    FROM tblGradeData JOIN tblCourseData ON tblGradeData.CourseID =
tblCourseData.CourseID

                    JOIN tblStudentData ON tblGradeData.StudID = tblStudentData.StudID"

            Using command As New SQLiteCommand(sqlQuery, connection)

                Dim reader As SQLiteDataReader = command.ExecuteReader()

                    'Write column headers to Excel
                    For colIndex As Integer = 0 To reader.FieldCount - 1
                        worksheet.Cells(1, colIndex + 1) = reader.GetName(colIndex)
                    Next
                    ' Write data to Excel
                    Dim rowIndex As Integer = 2
                    While reader.Read()
                        For colIndex As Integer = 0 To reader.FieldCount - 1
                            worksheet.Cells(rowIndex, colIndex + 1) = reader(colIndex).ToString()
                        Next
                        rowIndex += 1
                    End While
                reader.Close()
            End Using
            ' Save Excel file
            Dim excelFilePath As String = "Grades.xlsx"
            workbook.SaveAs(excelFilePath)
            ' Clean up
            ' After saving and before quitting
            workbook.Close(False)
            Marshal.ReleaseComObject(workbook)
            excelApp.Quit()
            Marshal.ReleaseComObject(excelApp)
```

```
        ' Release COM objects

        ReleaseComObject(worksheet)

        MessageBox.Show("Student Grades succesfully exported!")

    Else

        ' Code to execute when user chooses No or closes the prompt

    End If

End Sub

Private Sub ReleaseComObject(ByVal obj As Object)

    Try

        If obj IsNot Nothing Then

            System.Runtime.InteropServices.Marshal.ReleaseComObject(obj)

            obj = Nothing

        End If

    Catch ex As Exception

        obj = Nothing

    Finally

        GC.Collect()

    End Try

End Sub

'02. Excel Export Click : Ends

'03. PDF Export Click : Starts
```

```
   Private Sub btnExportToPDF_Click(sender As Object, e As EventArgs) Handles btnExportToPDF.Click

        ' Display a question prompt and get user's choice

        Dim userChoice As DialogResult = MessageBox.Show("Do you want to Export Student Grades in PDF Format?", "Question", MessageBoxButtons.YesNo, MessageBoxIcon.Question)

        ' Check the user's choice and perform actions accordingly

        If userChoice = DialogResult.Yes Then

            ' Code to execute when user chooses Yes

            ExportToPDF()

        End If

    End Sub

    ' Define a function to export to PDF

    Sub ExportToPDF()

        ' Create a new document

        Dim doc As New Document()

        Dim pdfFilePath As String = "C:\Users\Saidu\Desktop\Grades.pdf"

        Dim pdfWriter As PdfWriter = PdfWriter.GetInstance(doc, New FileStream(pdfFilePath, FileMode.Create))

        ' Open the document

        doc.Open()

        ' Query data from the database

        Dim query As String = "SELECT Grade, tblCourseData.CourseName, tblStudentData.FirstName, tblStudentData.LastName, tblStudentData.StudentID
```

```
                    FROM tblGradeData JOIN tblCourseData ON tblGradeData.CourseID =
tblCourseData.CourseID
                    JOIN tblStudentData ON tblGradeData.StudID = tblStudentData.StudID"
    Dim command As New SQLiteCommand(query, connection)
    Dim reader As SQLiteDataReader = command.ExecuteReader()
    ' Create a table in the PDF document
    Dim table As New PdfPTable(reader.FieldCount)
    While reader.Read()
      For i As Integer = 0 To reader.FieldCount - 1
        table.AddCell(reader(i).ToString())
      Next
    End While
    ' Add the table to the document
    doc.Add(table)
    ' Close the document
    doc.Close()
    connection.Close()
  End Sub
  '03. PDF Export Click : Ends
  'Search Button Logic Starts
  Public Sub ShowFilterResult(ProgramLevel As String, ProgramName As String, Year As
Integer, Semester As String)
```

```
Dim searchQuery As String = "SELECT DISTINCT GradeID, Grade, tblGradeData.Semester,
tblCourseData.CourseName, tblCourseData.Year, tblStudentData.FirstName,
tblStudentData.LastName, tblStudentData.StudentID, tblStudentData.ProgramLevel As ProgramLevel, tblStudentData.ProgramName As ProgramName
FROM tblGradeData JOIN tblCourseData ON tblGradeData.CourseID = tblCourseData.CourseID
JOIN tblStudentData ON tblGradeData.StudID = tblStudentData.StudID
WHERE tblStudentData.ProgramLevel LIKE '%' || @ProgramLevel || '%' AND tblStudentData.ProgramName LIKE '%' || @ProgramName || '%' AND tblStudentData.Year LIKE '%' || @Year || '%'  AND tblGradeData.Semester LIKE '%' || @Semester || '%'
ORDER BY CASE tblGradeData.Semester
WHEN 'First' THEN 1
WHEN 'Second' THEN 2
END"
Dim filterCmd As New SQLiteCommand(searchQuery, connection)
' Use parameterized query to avoid SQL injection
filterCmd.Parameters.AddWithValue("@ProgramLevel", ProgramLevel)
filterCmd.Parameters.AddWithValue("@ProgramName", ProgramName)
filterCmd.Parameters.AddWithValue("@Year", Year)
filterCmd.Parameters.AddWithValue("@Semester", Semester)
Dim resultAdapter As New SQLiteDataAdapter
resultAdapter.SelectCommand = filterCmd
' Create an instance of the DataTable class
```

```
        Dim resultDataTable As New Data.DataTable()

        resultAdapter.Fill(resultDataTable)

        dgvViewStudentGradeData.DataSource = resultDataTable

        incrementedVariable()

        '...

        dgvViewStudentGradeData.Columns("No").DisplayIndex = 0

        dgvViewStudentGradeData.Columns("FirstName").DisplayIndex = 1

        dgvViewStudentGradeData.Columns("LastName").DisplayIndex = 2

        dgvViewStudentGradeData.Columns("StudentID").DisplayIndex = 3

        dgvViewStudentGradeData.Columns("CourseName").DisplayIndex = 4

        dgvViewStudentGradeData.Columns("Grade").DisplayIndex = 5

        dgvViewStudentGradeData.Columns("Semester").DisplayIndex = 6

        dgvViewStudentGradeData.Columns("Edit").DisplayIndex = 7

        dgvViewStudentGradeData.Columns("Delete").DisplayIndex = 8

    End Sub

    Private Sub btnFilterSearchD_Click(sender As Object, e As EventArgs) Handles
btnFilterSearchD.Click

        Dim ProgramLevel As String = cmbSelectProgramLevel.SelectedItem

        Dim ProgramName As String = cmbSelectProgram.SelectedItem

        Dim Year As String = cmbSelectYear.SelectedItem

        Dim Semester As String = cmbSemester.SelectedItem

        ShowFilterResult(ProgramLevel, ProgramName, Year, Semester)

    End Sub
    Private Sub btnResetD_Click(sender As Object, e As EventArgs) Handles btnResetD.Click
        RefreshDataGridView()
        '...
        cmbSelectProgramLevel.SelectedIndex = 0
        cmbSelectProgram.SelectedIndex = 0
        cmbSelectYear.SelectedIndex = 0
        cmbSemester.SelectedIndex = 0
    End Sub
    'Search Button Logic Ends
    'Form Controls
    Private Sub btnBack_Click(sender As Object, e As EventArgs) Handles btnBack.Click
        frmDashboard.Show()
        Me.Close()
    End Sub
End Class
```

2. **Edit Individual Student Grade Logic**

```
Imports System.Data.SQLite
Public Class frmGradeUpdateData
    'Database Connections
    Private connection As SQLiteConnection
    Private command As SQLiteCommand
    'Initializing SQLite Database connections
    Public Sub New()
```

```
        InitializeComponent()

        Try

            connection = New SQLiteConnection("Data Source=gis.db;Version=3;Pooling=true;")

            connection.Open()

            command = connection.CreateCommand()

        Catch ex As Exception

            MessageBox.Show("Error connecting to the database: " & ex.Message, "Connection
Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

        End Try

    End Sub

    ' Properties to store the data

    Public Property GradeID As Integer

    Public Property FirstName As String

    Public Property LastName As String

    Public Property StudentID As Integer

    Public Property CourseName As String

    Public Property Grade As String

    Public Property Semester As String

    Public Property Year As String

    Public Property ProgramLevel As String

    Public Property ProgramName As String

    ' Method to set the data
    Public Sub SetData(column1Value As String, column2Value As String, column3Value As String, column4Value As String, column5Value As String, column6Value As String, column7Value As String, column8Value As String, column9Value As String, column10Value As String)

        GradeID = Integer.Parse(column1Value)

        FirstName = column2Value

        LastName = column3Value

        StudentID = Integer.Parse(column4Value)

        CourseName = column5Value

        Grade = column6Value

        Year = column7Value

        Semester = column8Value

        ProgramLevel = column9Value

        ProgramName = column10Value

    End Sub

    Public Sub GetCourseData(Year As Integer, ProgramLevel As String, ProgramName As String)

        Dim query As String = "SELECT * FROM tblCourseData WHERE Year = @Year AND ProgramLevel = @ProgramLevel AND ProgramName = @ProgramName"

        Dim command As New SQLiteCommand(query, connection)

        ' Use parameterized query to avoid SQL injection

        command.Parameters.AddWithValue("@Year", Year)

        command.Parameters.AddWithValue("@ProgramLevel", ProgramLevel)

        command.Parameters.AddWithValue("@ProgramName", ProgramName)
        Dim dataAdapter As New SQLiteDataAdapter
        dataAdapter.SelectCommand = command
        Dim DT As New DataTable()
        ' Fill DataTable with data
        dataAdapter.Fill(DT)
        ' Insert default row
        Dim row As DataRow = DT.NewRow
        row("CourseID") = 0
        row("CourseName") = "Select Course"
        DT.Rows.InsertAt(row, 0)
        cmbCourse.DataSource = DT
        cmbCourse.DisplayMember = "CourseName"
        cmbCourse.ValueMember = "CourseID"
    End Sub
    Private Sub frmGradeUpdateData_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        GetCourseData(Year, ProgramLevel, ProgramName)
        'Display the data in the labels, textboxes, etc. on the TargetForm
        txtGradeID.Text = GradeID
        txtFName.Text = FirstName
        txtLName.Text = LastName
        txtStudentID.Text = StudentID
        cmbCourse.SelectedItem = CourseName

        cmbSelectLetterGrade.SelectedItem = Grade

        cmbSemester.SelectedItem = Semester

        'To hide the textbox for all Unique IDs

        txtGradeID.Visible = False



        ' ...

    End Sub

    'Response Messages Starts

    Private Sub ShowSuccessMessage()

       MessageBox.Show("Student Grades updated successful!", "Success",
MessageBoxButtons.OK, MessageBoxIcon.Information)
    End Sub

    Private Sub ShowFailureMessage()

        MessageBox.Show("Student Grades update unsuccessful!", "Failure",
MessageBoxButtons.OK, MessageBoxIcon.Information)
    End Sub

    'Response Messages Ends

    'Creating the CRUD METHODS - STARTS

    '01. Get Method

    Private Function GetData() As Data.DataTable

       command.CommandText = "SELECT GradeID, Grade, tblGradeData.Semester,
tblCourseData.CourseName, tblCourseData.Year, tblStudentData.FirstName,
tblStudentData.LastName, tblStudentData.StudentID
        FROM    tblGradeData    JOIN    tblCourseData    ON    tblGradeData.CourseID    =
tblCourseData.CourseID

        JOIN tblStudentData ON tblGradeData.StudID = tblStudentData.StudID"
        Dim adapter As New SQLiteDataAdapter(command)
        Dim dataTable As New Data.DataTable()
        adapter.Fill(dataTable)
        Return dataTable
    End Function
    'Auto Data Refresh Method - STARTS
    Private Sub RefreshDataGridView()
        frmGradeModify.dgvViewStudentGradeData.DataSource = GetData()
        incrementedVariable()
        frmGradeModify.dgvViewStudentGradeData.Columns("No").DisplayIndex = 0
        frmGradeModify.dgvViewStudentGradeData.Columns("FirstName").DisplayIndex = 1
        frmGradeModify.dgvViewStudentGradeData.Columns("LastName").DisplayIndex = 2
        frmGradeModify.dgvViewStudentGradeData.Columns("StudentID").DisplayIndex = 3
        frmGradeModify.dgvViewStudentGradeData.Columns("CourseName").DisplayIndex = 4
        frmGradeModify.dgvViewStudentGradeData.Columns("Grade").DisplayIndex = 5
        frmGradeModify.dgvViewStudentGradeData.Columns("Semester").DisplayIndex = 6
        frmGradeModify.dgvViewStudentGradeData.Columns("Edit").DisplayIndex = 7
        frmGradeModify.dgvViewStudentGradeData.Columns("Delete").DisplayIndex = 8
    End Sub
    'Auto Data Refresh Method - ENDS
    'Incremented Variable Starts
    Public Sub incrementedVariable()
```

```
        ' Get the number of rows in the DataGridView
        Dim rowCount As Integer = frmGradeModify.dgvViewStudentGradeData.Rows.Count
        ' Loop through each row and assign incremented values to the "Number" column
        For i As Integer = 0 To rowCount - 1
            frmGradeModify.dgvViewStudentGradeData.Rows(i).Cells("No").Value = i + 1
        Next
    End Sub
    'Incremented Variable Ends
    '02. Update Method
    Private Sub UpdateData(CourseID As Integer, GradeID As Integer, Grade As String, Semester
As String)
        command.CommandText = "UPDATE tblGradeData SET Grade = @Grade, CourseID =
@CourseID, Semester = @Semester WHERE GradeID = @GradeID"
        command.Parameters.AddWithValue("@CourseID", CourseID)
        command.Parameters.AddWithValue("@GradeID", GradeID)
        command.Parameters.AddWithValue("@Grade", Grade)
        command.Parameters.AddWithValue("@Semester", Semester)
        If (command.ExecuteNonQuery() > 0) Then
            ShowSuccessMessage()
        Else
            ShowFailureMessage()
        End If
    End Sub

    'Clear Data Method - STARTS

    Private Sub ClearData()

        txtFName.Text = " "

        txtLName.Text = " "

        txtStudentID.Text = " "

        cmbSelectLetterGrade.SelectedItem = Nothing

        cmbCourse.SelectedItem = Nothing

        cmbSemester.SelectedItem = Nothing

    End Sub

    'Clear Data Method - ENDS

    Private Sub btnUpdateData_Click(sender As Object, e As EventArgs) Handles
btnUpdateData.Click

        Dim CourseID As Integer = cmbCourse.SelectedValue

        Dim GradeID As Integer = txtGradeID.Text

        Dim Grade As String = cmbSelectLetterGrade.SelectedItem

        Dim Semester As String = cmbSemester.SelectedItem

        UpdateData(CourseID, GradeID, Grade, Semester)

        ClearData()

        RefreshDataGridView()

    End Sub
    'Form Control
    Private Sub btnBack_Click(sender As Object, e As EventArgs) Handles btnBack.Click
        frmGradeModify.Show()
        Me.Close()
    End Sub
End Class
```

(c) View all Student Grade

```
Imports System.Data.SQLite
Public Class frmViewAllStudentGrade
    'Database Connections
    Private connection As SQLiteConnection
    Private command As SQLiteCommand
    'Initializing SQLite Database connections
    Public Sub New()
        InitializeComponent()
        Try
            connection = New SQLiteConnection("Data Source=gis.db;Version=3;Pooling=true;")
            connection.Open()
            command = connection.CreateCommand()
        Catch ex As Exception
            MessageBox.Show("Error connecting to the database: " & ex.Message, "Connection
Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
        End Try
    End Sub

    Private Sub frmStudentGradeSummary_Load(sender As Object, e As EventArgs) Handles
MyBase.Load

        RefreshDataGridView()

        '...

        cmbSelectProgramLevel.SelectedIndex = 0

        cmbSelectProgram.SelectedIndex = 0

        cmbSelectYear.SelectedIndex = 0

    End Sub

    'Response Messages Starts

    Private Sub ShowSuccessMessage()

        MessageBox.Show("Grade Data successful removed!", "Success", MessageBoxButtons.OK,
MessageBoxIcon.Information)
    End Sub

    Private Sub ShowFailureMessage()

        MessageBox.Show("Failed to remove Grade Data unsuccessful!", "Failure",
MessageBoxButtons.OK, MessageBoxIcon.Information)
    End Sub

    Private Sub ShowSuccessMessageImport()
        MessageBox.Show("Course Data successful Imported!", "Success",
MessageBoxButtons.OK, MessageBoxIcon.Information)
    End Sub

    Private Sub ShowFailureMessageImport()

        MessageBox.Show("Failed to Import Course Data!", "Failure", MessageBoxButtons.OK,
MessageBoxIcon.Information)

    End Sub
    'Response Messages Ends

    'Creating the CRUD METHODS - STARTS

    '01. Get Method

    Private Function GetData() As Data.DataTable

        command.CommandText = "SELECT DISTINCT tblStudentData.FirstName,
tblStudentData.LastName, tblStudentData.StudentID, tblStudentData.ProgramLevel,
tblStudentData.ProgramName, tblStudentData.Year FROM tblGradeData JOIN tblCourseData
ON tblGradeData.CourseID = tblCourseData.CourseID JOIN tblStudentData ON
tblGradeData.StudID = tblStudentData.StudID"
        Dim adapter As New SQLiteDataAdapter(command)

        Dim dataTable As New Data.DataTable()

        adapter.Fill(dataTable)

        Return dataTable

    End Function

    'Auto Data Refresh Method - STARTS

    Private Sub RefreshDataGridView()

        dgvViewAllGrades.DataSource = GetData()

        incrementedVariable()

        dgvViewAllGrades.Columns("No").DisplayIndex = 0

        dgvViewAllGrades.Columns("FirstName").DisplayIndex = 1

        dgvViewAllGrades.Columns("LastName").DisplayIndex = 2

        dgvViewAllGrades.Columns("StudentID").DisplayIndex = 3

        dgvViewAllGrades.Columns("ProgramLevel").DisplayIndex = 4

        dgvViewAllGrades.Columns("ProgramName").DisplayIndex = 5
        dgvViewAllGrades.Columns("Year").DisplayIndex = 6
        dgvViewAllGrades.Columns("btnViewGrades").DisplayIndex = 7
    End Sub
    'Auto Data Refresh Method - ENDS
    'Incremented Variable Starts
    Public Sub incrementedVariable()
        ' Get the number of rows in the DataGridView
        Dim rowCount As Integer = dgvViewAllGrades.Rows.Count
        ' Loop through each row and assign incremented values to the "Number" column
        For i As Integer = 0 To rowCount - 1
            dgvViewAllGrades.Rows(i).Cells("No").Value = i + 1
        Next
    End Sub
    'Search Button Logic Starts
    Public Sub ShowFilterResult(ProgramLevel As String, ProgramName As String, Year As
Integer)
        Dim searchQuery As String = "SELECT DISTINCT tblStudentData.FirstName,
tblStudentData.LastName, tblStudentData.StudentID,
        tblStudentData.ProgramLevel, tblStudentData.ProgramName, tblStudentData.Year FROM
tblGradeData JOIN tblCourseData
        ON tblGradeData.CourseID = tblCourseData.CourseID
        JOIN tblStudentData ON tblGradeData.StudID = tblStudentData.StudID
        WHERE tblStudentData.ProgramLevel LIKE '%' || @ProgramLevel || '%' AND
tblStudentData.ProgramName LIKE '%' || @ProgramName || '%' AND tblStudentData.Year
LIKE '%' || @Year || '%'"

        Dim filterCmd As New SQLiteCommand(searchQuery, connection)

        ' Use parameterized query to avoid SQL injection

        filterCmd.Parameters.AddWithValue("@ProgramLevel", ProgramLevel)

        filterCmd.Parameters.AddWithValue("@ProgramName", ProgramName)

        filterCmd.Parameters.AddWithValue("@Year", Year)

        Dim resultAdapter As New SQLiteDataAdapter

        resultAdapter.SelectCommand = filterCmd

        ' Create an instance of the DataTable class

        Dim resultDataTable As New Data.DataTable()

        resultAdapter.Fill(resultDataTable)

        dgvViewAllGrades.DataSource = resultDataTable

        incrementedVariable()

    End Sub

    Private Sub btnFilterSearchE_Click(sender As Object, e As EventArgs) Handles
btnFilterSearchE.Click

        Dim ProgramLevel As String = cmbSelectProgramLevel.SelectedItem

        Dim ProgramName As String = cmbSelectProgram.SelectedItem

        Dim Year As String = cmbSelectYear.SelectedItem

        ShowFilterResult(ProgramLevel, ProgramName, Year)
    End Sub

    Private Sub btnResetE_Click(sender As Object, e As EventArgs) Handles btnResetE.Click

        RefreshDataGridView()

        '...

        cmbSelectProgramLevel.SelectedIndex = 0

        cmbSelectProgram.SelectedIndex = 0

        cmbSelectYear.SelectedIndex = 0

    End Sub

    'Filter functionality ends here

    'Creating the CRUD METHODS - ENDS

    'Handling User Interaction

    '01. Create Click : Starts with Datagrid Cellcontent click event

    Private Sub dgvStudentGradeSummary_CellContentClick(sender As Object, e As
DataGridViewCellEventArgs) Handles dgvViewAllGrades.CellContentClick

        ' Check if the clicked cell is in the button column

        If e.ColumnIndex = 1 AndAlso e.RowIndex >= 0 Then

            ' Perform action when button is clicked

            Dim selectedRow As DataGridViewRow = dgvViewAllGrades.Rows(e.RowIndex)

            'Get the data from the cells

            Dim column1Value As String = selectedRow.Cells("StudentID").Value.ToString()

            ' Create an instance of the target form

            Dim targetForm As New frmStudentIndividualGradesSummaryShow()
            ' Pass the data to the target form
            targetForm.SetData(column1Value)
            ' Show the target form
            targetForm.Show()
        End If
    End Sub
    '01. Create Click : Ends with Datagrid Cellcontent click event
    'Form Controls
    Private Sub btnBack_Click(sender As Object, e As EventArgs) Handles btnBack.Click
        frmDashboard.Show()
        Me.Close()
    End Sub
End Class
```

1. **View Individual Student Grade Logic**

```
Imports System.Data.SQLite
Public Class frmStudentIndividualGradesSummaryShow
    'Database Connections
    Private connection As SQLiteConnection
    Private command As SQLiteCommand
    'Initializing SQLite Database connections
    Public Sub New()
        InitializeComponent()
        Try
```

```
            connection = New SQLiteConnection("Data Source=gis.db;Version=3;Pooling=true;")
            connection.Open()
            command = connection.CreateCommand()
        Catch ex As Exception
            MessageBox.Show("Error connecting to the database: " & ex.Message, "Connection
Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
        End Try
    End Sub
    ' Properties to store the data
    Public Property StudentID As Integer
    Public Property academicYear As Integer = DateTime.Now.Year 'Get the current year as
Academic year
    Public Property joinYear As Integer = joinYear
    ' Method to set the data
    Public Sub SetData(column1Value As String)
        StudentID = Integer.Parse(column1Value)
    End Sub
    Private Sub frmStudentIndividualGradesSummaryShow_Load(sender As Object, e As
EventArgs) Handles MyBase.Load
        CreateDatabaseTable()
        GetStudentPersonalData(StudentID)
        RefreshStudentGradesDataGridView()
        cmbSelectYear.SelectedIndex = 0

        'Peform student automatic update

        UpdateStudentYear(StudentID, academicYear, joinYear)

    End Sub

    'Creating the CRUD METHODS - STARTS

    '01. Get Method

    Private Sub GetStudentPersonalData(StudentID As Integer)

        Dim query As String = "SELECT Grade, tblGradeData.Semester, 
tblCourseData.CourseName, tblCourseData.Year, tblStudentData.FirstName, 
tblStudentData.LastName,
        tblStudentData.StudentID,tblStudentData.ProgramLevel, tblStudentData.ProgramName, 
tblStudentData.Year FROM tblGradeData JOIN tblCourseData ON tblGradeData.CourseID = 
tblCourseData.CourseID
        JOIN tblStudentData ON tblGradeData.StudID = tblStudentData.StudID WHERE 
tblStudentData.StudentID = @StudentID ORDER BY tblGradeData.Semester ASC"
        Dim command As New SQLiteCommand(query, connection)

        command.Parameters.AddWithValue("@StudentID", StudentID) ' Pass the StudentID 
parameter

        Dim reader As SQLiteDataReader = command.ExecuteReader()

        While reader.Read()

            Dim value1 As String = reader.GetString(4)

            Dim value2 As String = reader.GetString(5)

            Dim value3 As Integer = reader.GetInt32(6)

            Dim value4 As String = reader.GetString(7)

            Dim value5 As String = reader.GetString(8)

            Dim value6 As Integer = reader.GetInt32(9)
```

```
         grpbGradeData.Text = value1 & " " & value2
         lblProgram.Text = value4 & " " & value5 & " " & "Year " & " " & value6
         lblStudentID.Text = value3
      End While
      reader.Close()
   End Sub
   Private Function GetStudentGradeData(ByVal StudentID As Integer) As Data.DataTable
      command.CommandText = "SELECT DISTINCT GradeID, Grade, tblGradeData.Semester,
tblCourseData.CourseName, tblCourseData.CreditHour
 FROM    tblGradeData    JOIN    tblCourseData    ON    tblGradeData.CourseID    =
tblCourseData.CourseID
                 JOIN tblStudentData ON tblGradeData.StudID = tblStudentData.StudID
                 WHERE tblStudentData.StudentID = @StudentID
                 ORDER BY CASE tblGradeData.Semester
                    WHEN 'First' THEN 1
                    WHEN 'Second' THEN 2
                    END"
      command.Parameters.AddWithValue("@StudentID", StudentID)
      Dim adapter As New SQLiteDataAdapter(command)
      Dim dataTable As New Data.DataTable()
      adapter.Fill(dataTable)
      Return dataTable
```

```
End Function

'Auto Data Refresh Method - STARTS

Private Sub RefreshStudentGradesDataGridView()

    dgvViewStudentSingleGradeData.DataSource = GetStudentGradeData(StudentID)

    incrementedVariable()

    dgvViewStudentSingleGradeData.Columns("No").DisplayIndex = 0

    dgvViewStudentSingleGradeData.Columns("CourseName").DisplayIndex = 1

    dgvViewStudentSingleGradeData.Columns("Grade").DisplayIndex = 2

    dgvViewStudentSingleGradeData.Columns("Semester").DisplayIndex = 3

    dgvViewStudentSingleGradeData.Columns("GradeWeight").DisplayIndex = 4

    ' Calculate SGPA and display

    CalculateAndDisplaySGPA()

End Sub

'Auto Data Refresh Method - ENDS

'Incremented Variable Starts

Public Sub incrementedVariable()

    ' Get the number of rows in the DataGridView

    Dim rowCount As Integer = dgvViewStudentSingleGradeData.Rows.Count

    ' Loop through each row and assign incremented values to the "Number" column

    For i As Integer = 0 To rowCount - 1

        dgvViewStudentSingleGradeData.Rows(i).Cells("No").Value = i + 1

    Next
```

```
End Sub
'Incremented Variable Ends
'02. Calculating the SGPA for the student Starts
Private Sub CalculateAndDisplaySGPA()
    Dim totalCreditHours As Integer = 0
    Dim totalGW As Double = 0
    For Each row As DataGridViewRow In dgvViewStudentSingleGradeData.Rows
        Dim letterGrade As String = row.Cells("Grade").Value.ToString()
        Dim creditHours As Integer = Convert.ToInt32(row.Cells("CreditHour").Value)
        Dim numericalWeight As Double = GetNumericalWeight(letterGrade)
        Dim gradeWeight As Double = numericalWeight * creditHours
        row.Cells("GradeWeight").Value = gradeWeight
        totalGW += gradeWeight
        totalCreditHours += creditHours
    Next
    Dim finalSGPA As Double = totalGW / totalCreditHours
    ' Display the final Data in selected controls
    lblSGPA.Text = finalSGPA.ToString("N2")
    lblTotalCreditHour.Text = totalCreditHours
    If finalSGPA >= 3.0 Then
        lblCommentValue.Text = "Pass"
    ElseIf finalSGPA < 3.0 Then
        lblCommentValue.Text = "Fail"
    Else
        lblCommentValue.Text = "Student Status"
    End If
End Sub


' Helper function to convert letter grades to numerical weight values
Private Function GetNumericalWeight(letterGrade As String) As Double
    ' Defining mapping from letter grades to numerical values
    Select Case letterGrade.ToUpper()
        Case "A"
            Return 5.0
        Case "B"
            Return 4.0
        Case "C"
            Return 3.0
        Case "D"
            Return 2.0
        Case "E"
            Return 1.0
        Case "F"
            Return 0.0
        Case Else
            Return 0.0
    End Select
  End Function
  '02. Calculating the SGPA for the Student Ends
  'Create database table If Not exist
  Private Sub CreateDatabaseTable()
    command.CommandText = "CREATE TABLE IF NOT EXISTS tblStoreAllGrades (
AllGradesID INTEGER PRIMARY KEY AUTOINCREMENT,
Grade1 INTEGER,
Grade2 INTEGER,
Grade3 INTEGER,
Grade4 INTEGER,
Grade5 INTEGER,
Grade6 INTEGER,
Grade7 INTEGER,
Grade8 INTEGER,
Grade9 INTEGER,
Grade10 INTEGER,
Grade11 INTEGER,
Grade12 INTEGER,
Grade13 INTEGER,
Grade14 INTEGER,
TotalCreditHours INTEGER,
SGPA REAL,
GradeComment TEXT,

FOREIGN KEY (Grade1) REFERENCES tblGradeData(GradeID),

FOREIGN KEY (Grade2) REFERENCES tblGradeData(GradeID),

FOREIGN KEY (Grade3) REFERENCES tblGradeData(GradeID),

FOREIGN KEY (Grade4) REFERENCES tblGradeData(GradeID),

FOREIGN KEY (Grade5) REFERENCES tblGradeData(GradeID),

FOREIGN KEY (Grade6) REFERENCES tblGradeData(GradeID),

FOREIGN KEY (Grade7) REFERENCES tblGradeData(GradeID),

FOREIGN KEY (Grade8) REFERENCES tblGradeData(GradeID),

FOREIGN KEY (Grade9) REFERENCES tblGradeData(GradeID),

FOREIGN KEY (Grade10) REFERENCES tblGradeData(GradeID),

FOREIGN KEY (Grade11) REFERENCES tblGradeData(GradeID),

FOREIGN KEY (Grade12) REFERENCES tblGradeData(GradeID),

FOREIGN KEY (Grade13) REFERENCES tblGradeData(GradeID),

FOREIGN KEY (Grade14) REFERENCES tblGradeData(GradeID))"

      command.ExecuteNonQuery()

  End Sub

  'Response Messages Starts

  Private Sub ShowSuccessMessage()

      MessageBox.Show("Student All Grades Data successful Saved!", "Success",
MessageBoxButtons.OK, MessageBoxIcon.Information)

  End Sub
```

```
Private Sub ShowFailureMessage()
    MessageBox.Show("Fail to save student grades!", "Failure", MessageBoxButtons.OK, MessageBoxIcon.Information)
End Sub
Private Sub ShowUpdateSuccessMessage()
    MessageBox.Show("Student All Grades Data successful Updated!", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information)
End Sub
Private Sub ShowUpdateFailureMessage()
    MessageBox.Show("Fail to update student grades!", "Failure", MessageBoxButtons.OK, MessageBoxIcon.Information)
End Sub
'Response Messages Ends
'01. CRUD METHODS Starts
Public Sub insertStudentGrade(grade1 As String, grade2 As String, grade3 As String, grade4 As String, grade5 As String, grade6 As String, grade7 As String, grade8 As String, grade9 As String, grade10 As String, grade11 As String, grade12 As String, grade13 As String, grade14 As String, TotalCreditHours As String, SGPA As String, GradeComment As String)
    Dim insertQuery As String = "INSERT INTO tblStoreAllGrades (Grade1, Grade2, Grade3, Grade4, Grade5, Grade6, Grade7,Grade8, Grade9, Grade10, Grade11, Grade12, Grade13, Grade14, TotalCreditHours, SGPA, GradeComment) VALUES (@Grade1, @Grade2, @Grade3, @Grade4, @Grade5, @Grade6, @Grade7, @Grade8, @Grade9, @Grade10, @Grade11, @Grade12, @Grade13, @Grade14, @TotalCreditHours, @SGPA, @GradeComment)"
    Dim insertCommand As New SQLiteCommand(insertQuery, connection)
    insertCommand.Parameters.AddWithValue("@Grade1", grade1)
    insertCommand.Parameters.AddWithValue("@Grade2", grade2)
    insertCommand.Parameters.AddWithValue("@Grade3", grade3)
    insertCommand.Parameters.AddWithValue("@Grade4", grade4)
    insertCommand.Parameters.AddWithValue("@Grade5", grade5)
    insertCommand.Parameters.AddWithValue("@Grade6", grade6)
    insertCommand.Parameters.AddWithValue("@Grade7", grade7)
    insertCommand.Parameters.AddWithValue("@Grade8", grade8)
    insertCommand.Parameters.AddWithValue("@Grade9", grade9)
    insertCommand.Parameters.AddWithValue("@Grade10", grade10)
    insertCommand.Parameters.AddWithValue("@Grade11", grade11)
    insertCommand.Parameters.AddWithValue("@Grade12", grade12)
    insertCommand.Parameters.AddWithValue("@Grade13", grade13)
    insertCommand.Parameters.AddWithValue("@Grade14", grade14)
    insertCommand.Parameters.AddWithValue("@TotalCreditHours", TotalCreditHours)
    insertCommand.Parameters.AddWithValue("@SGPA", SGPA)
    insertCommand.Parameters.AddWithValue("@GradeComment", GradeComment)
    If (insertCommand.ExecuteNonQuery() > 0) Then
        ShowSuccessMessage()
    Else
        ShowFailureMessage()
    End If
End Sub
```

```
Public Sub updateStudentGrade(grade1 As String, grade2 As String, grade3 As String, grade4 As String, grade5 As String, grade6 As String, grade7 As String, grade8 As String, grade9 As String, grade10 As String, grade11 As String, grade12 As String, grade13 As String, grade14 As String, TotalCreditHours As String, SGPA As String, GradeComment As String, AllGradesID As Integer)

    Dim insertQuery As String = "UPDATE tblStoreAllGrades SET Grade1 = @Grade1, Grade2 = @Grade2, Grade3 = @Grade3, Grade4 = @Grade4, Grade5 = @Grade5, Grade6 = @Grade6, Grade7 = @Grade7, Grade8 = @Grade8, Grade9 = @Grade9, Grade10 = @Grade10, Grade11 = @Grade11, Grade12 = @Grade12, Grade13 = @Grade13, Grade14 = @Grade14, TotalCreditHours = @TotalCreditHours, SGPA = @SGPA, GradeComment = @GradeComment WHERE AllGradesID = @AllGradesID"

    Dim insertCommand As New SQLiteCommand(insertQuery, connection)

    insertCommand.Parameters.AddWithValue("@Grade1", grade1)

    insertCommand.Parameters.AddWithValue("@Grade2", grade2)

    insertCommand.Parameters.AddWithValue("@Grade3", grade3)

    insertCommand.Parameters.AddWithValue("@Grade4", grade4)

    insertCommand.Parameters.AddWithValue("@Grade5", grade5)

    insertCommand.Parameters.AddWithValue("@Grade6", grade6)

    insertCommand.Parameters.AddWithValue("@Grade7", grade7)

    insertCommand.Parameters.AddWithValue("@Grade8", grade8)

    insertCommand.Parameters.AddWithValue("@Grade9", grade9)

    insertCommand.Parameters.AddWithValue("@Grade10", grade10)

    insertCommand.Parameters.AddWithValue("@Grade11", grade11)

    insertCommand.Parameters.AddWithValue("@Grade12", grade12)

    insertCommand.Parameters.AddWithValue("@Grade13", grade13)
```

```
        insertCommand.Parameters.AddWithValue("@Grade14", grade14)

        insertCommand.Parameters.AddWithValue("@TotalCreditHours", TotalCreditHours)

        insertCommand.Parameters.AddWithValue("@SGPA", SGPA)

        insertCommand.Parameters.AddWithValue("@GradeComment", GradeComment)

        insertCommand.Parameters.AddWithValue("@AllGradesID", AllGradesID)

        If (insertCommand.ExecuteNonQuery() > 0) Then

            ShowUpdateSuccessMessage()

        Else

            ShowUpdateFailureMessage()

        End If

    End Sub

    '01. CRUD METHODS Ends

    Private Sub btnSaveGrade_Click(sender As Object, e As EventArgs) Handles
btnSaveGrade.Click

        If dgvViewStudentSingleGradeData.RowCount >= 10 Then

           Dim grade1 As String =
dgvViewStudentSingleGradeData.Rows(0).Cells("GradeID").Value.ToString()
            Dim grade2 As String =
dgvViewStudentSingleGradeData.Rows(1).Cells("GradeID").Value.ToString()
            Dim grade3 As String =
dgvViewStudentSingleGradeData.Rows(2).Cells("GradeID").Value.ToString()
            Dim grade4 As String =
dgvViewStudentSingleGradeData.Rows(3).Cells("GradeID").Value.ToString()
            Dim grade5 As String =
dgvViewStudentSingleGradeData.Rows(4).Cells("GradeID").Value.ToString()
            Dim grade6 As String =
dgvViewStudentSingleGradeData.Rows(5).Cells("GradeID").Value.ToString()
            Dim grade7 As String =
dgvViewStudentSingleGradeData.Rows(6).Cells("GradeID").Value.ToString()
            Dim grade8 As String =
dgvViewStudentSingleGradeData.Rows(7).Cells("GradeID").Value.ToString()
```

```
            Dim grade9 As String =
dgvViewStudentSingleGradeData.Rows(8).Cells("GradeID").Value.ToString()
            Dim grade10 As String =
dgvViewStudentSingleGradeData.Rows(9).Cells("GradeID").Value.ToString()
            Dim grade11 As String =
dgvViewStudentSingleGradeData.Rows(10).Cells("GradeID").Value.ToString()
            Dim grade12 As String =
dgvViewStudentSingleGradeData.Rows(11).Cells("GradeID").Value.ToString()
            Dim grade13 As String =
dgvViewStudentSingleGradeData.Rows(10).Cells("GradeID").Value.ToString()
            Dim grade14 As String =
dgvViewStudentSingleGradeData.Rows(11).Cells("GradeID").Value.ToString()

            Dim totalCreditHours As Integer = Integer.Parse(lblTotalCreditHour.Text)

            Dim sgpa As Double = Double.Parse(lblSGPA.Text)

            Dim gradeComment As String = lblCommentValue.Text

            ' Display a question prompt and get user's choice

            Dim userChoice As DialogResult = MessageBox.Show("Do you want to save the final
grades for this Student?", "Insert Student Final Grades", MessageBoxButtons.YesNo,
MessageBoxIcon.Question)

            ' Check the user's choice and perform actions accordingly

            If userChoice = DialogResult.Yes Then

                ' Code to execute when user chooses Yes

                insertStudentGrade(grade1, grade2, grade3, grade4, grade5, grade6, grade7, grade8,
grade9, grade10, grade11, grade12, grade13, grade14, totalCreditHours, sgpa, gradeComment)

            End If

        Else

            ' There are fewer Grades available in the DataGridView

            MessageBox.Show("There are fewer Grades available at the moment and cannot be
added into the database just yet!.")
```

```
        End If

    End Sub

    Private Sub btnUpdateGrade_Click(sender As Object, e As EventArgs) Handles
btnUpdateGrade.Click

        If dgvViewStudentSingleGradeData.RowCount >= 10 Then

            Dim grade1 As String =
dgvViewStudentSingleGradeData.Rows(0).Cells("GradeID").Value.ToString()
            Dim grade2 As String =
dgvViewStudentSingleGradeData.Rows(1).Cells("GradeID").Value.ToString()
            Dim grade3 As String =
dgvViewStudentSingleGradeData.Rows(2).Cells("GradeID").Value.ToString()
            Dim grade4 As String =
dgvViewStudentSingleGradeData.Rows(3).Cells("GradeID").Value.ToString()
            Dim grade5 As String =
dgvViewStudentSingleGradeData.Rows(4).Cells("GradeID").Value.ToString()
            Dim grade6 As String =
dgvViewStudentSingleGradeData.Rows(5).Cells("GradeID").Value.ToString()
            Dim grade7 As String =
dgvViewStudentSingleGradeData.Rows(6).Cells("GradeID").Value.ToString()
            Dim grade8 As String =
dgvViewStudentSingleGradeData.Rows(7).Cells("GradeID").Value.ToString()
            Dim grade9 As String =
dgvViewStudentSingleGradeData.Rows(8).Cells("GradeID").Value.ToString()
            Dim grade10 As String =
dgvViewStudentSingleGradeData.Rows(9).Cells("GradeID").Value.ToString()
            Dim grade11 As String =
dgvViewStudentSingleGradeData.Rows(10).Cells("GradeID").Value.ToString()
            Dim grade12 As String =
dgvViewStudentSingleGradeData.Rows(11).Cells("GradeID").Value.ToString()
            Dim grade13 As String =
dgvViewStudentSingleGradeData.Rows(12).Cells("GradeID").Value.ToString()
            Dim grade14 As String =
dgvViewStudentSingleGradeData.Rows(13).Cells("GradeID").Value.ToString()

            Dim totalCreditHours As Integer = Integer.Parse(lblTotalCreditHour.Text)

            Dim sgpa As Double = Double.Parse(lblSGPA.Text)

            Dim gradeComment As String = lblCommentValue.Text
            Dim AllGradesID As Integer =
dgvViewStudentSingleGradeData.Rows(13).Cells("GradeID").Value.ToString()
            ' Display a question prompt and get user's choice

            Dim userChoice As DialogResult = MessageBox.Show("Do you want to update the final
grades for this Student?", "Update Final Grades", MessageBoxButtons.YesNo,
MessageBoxIcon.Question)

            ' Check the user's choice and perform actions accordingly

            If userChoice = DialogResult.Yes Then

                ' Code to execute when user chooses Yes

                updateStudentGrade(grade1, grade2, grade3, grade4, grade5, grade6, grade7, grade8,
grade9, grade10, grade11, grade12, grade13, grade14, totalCreditHours, sgpa, gradeComment,
StudentID)

            End If

        Else

            ' There are fewer Grades available in the DataGridView

            MessageBox.Show("There are fewer Grades available at the moment and cannot be
added into the database just yet!.")

        End If

    End Sub

    '03. Filter Button Logic Starts

    Public Sub ShowFilterResult(StudentID As Integer, Year As Integer)

        Dim searchQuery As String = "SELECT DISTINCT GradeID, Grade,
tblGradeData.Semester, tblCourseData.CourseName, tblCourseData.CreditHour

                        FROM tblGradeData

                        JOIN tblCourseData ON tblGradeData.CourseID = tblCourseData.CourseID
                        JOIN tblStudentData ON tblGradeData.StudID = tblStudentData.StudID
                        WHERE tblStudentData.StudentID = @StudentID AND tblCourseData.Year =
@Year
                        ORDER BY CASE tblGradeData.Semester
                          WHEN 'First' THEN 1
                          WHEN 'Second' THEN 2
                          END"
        Dim filterCmd As New SQLiteCommand(searchQuery, connection)
        ' Use parameterized query to avoid SQL injection
        filterCmd.Parameters.AddWithValue("@StudentID", StudentID)
        filterCmd.Parameters.AddWithValue("@Year", Year)
        Dim resultAdapter As New SQLiteDataAdapter
        resultAdapter.SelectCommand = filterCmd
        ' Create an instance of the DataTable class
        Dim resultDataTable As New Data.DataTable()
        resultAdapter.Fill(resultDataTable)
        dgvViewStudentSingleGradeData.DataSource = resultDataTable
        incrementedVariable()
        ' Set the display index of the columns
        dgvViewStudentSingleGradeData.Columns("No").DisplayIndex = 0
        dgvViewStudentSingleGradeData.Columns("CourseName").DisplayIndex = 1
        dgvViewStudentSingleGradeData.Columns("Grade").DisplayIndex = 2
        dgvViewStudentSingleGradeData.Columns("Semester").DisplayIndex = 3
        dgvViewStudentSingleGradeData.Columns("GradeWeight").DisplayIndex = 4
```

```
        ' Calculate SGPA and display
        CalculateAndDisplaySGPA()
    End Sub
    Private Sub btnFilterSearchF_Click(sender As Object, e As EventArgs) Handles
btnFilterSearchF.Click
        Dim StudentIDvalue As Integer = StudentID
        Dim Year As String = cmbSelectYear.SelectedItem
        ShowFilterResult(StudentIDvalue, Year)
    End Sub
    Private Sub btnResetF_Click(sender As Object, e As EventArgs) Handles btnResetF.Click
        RefreshStudentGradesDataGridView()
        '...
        cmbSelectYear.SelectedIndex = 0
    End Sub
    '03. Filter Button Logic Ends
    '... Student automatic year functions
    '04. Function to calculate the current level based on the academic year
    Private Sub UpdateData(StudentID As Integer, Year As Integer)
        command.CommandText = "UPDATE tblStudentData SET Year = @Year WHERE
StudentID = @StudentID"
        command.Parameters.AddWithValue("@StudentID", StudentID)
        command.Parameters.AddWithValue("@Year", Year)
        command.ExecuteNonQuery()
```

```
End Sub

Function CalculateStudentYear(academicYear As Integer, joinYear As Integer) As Integer

   Dim currentYear As Integer = DateTime.Now.Year

   Dim studentYear As Integer = currentYear - joinYear

   ' Adjust the level if the academic year is ahead of the current year

   If academicYear > currentYear Then

      studentYear -= academicYear - currentYear

   End If

   Return studentYear

End Function

'05. Update the student's level (Year) in the database

Sub UpdateStudentYear(StudentID As Integer, academicYear As Integer, joinYear As Integer)

   Dim studentYear As Integer = CalculateStudentYear(academicYear, joinYear)

   If lblCommentValue.Text = "Pass" AndAlso

   dgvViewStudentSingleGradeData.RowCount >= 10 Then

      UpdateData(StudentID, studentYear)

   Else

      'Do nothing

   End If

   ' ...

End Sub


'Form Controls
```

```
    Private Sub btnBack_Click(sender As Object, e As EventArgs) Handles btnBack.Click
        frmViewAllStudentGrade.Show()
        Me.Close()
    End Sub
End Class
```

(d) Summary (Shows the entire Grade Book of all students)

```
Imports System.Data.SQLite
Public Class frmViewGradebook
    'Database Connections
    Private connection As SQLiteConnection
    Private command As SQLiteCommand
    'Initializing SQLite Database connections
    Public Sub New()
        InitializeComponent()
        Try
            connection = New SQLiteConnection("Data Source=gis.db;Version=3;Pooling=true;")
            connection.Open()
            command = connection.CreateCommand()
        Catch ex As Exception
            MessageBox.Show("Error connecting to the database: " & ex.Message, "Connection
Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
        End Try
    End Sub

    Private Sub frmViewGradebook_Load(sender As Object, e As EventArgs) Handles
MyBase.Load

        RefreshDataGridView()

        '...

        cmbSelectProgramLevel.SelectedIndex = 0

        cmbSelectProgram.SelectedIndex = 0

        cmbSelectYear.SelectedIndex = 0

        cmbSemester.SelectedIndex = 0

        'Reset the top description

        lblProgram.Text = "All Students"

        lblSemester.Text = "First Semester"

    End Sub

    'Response Messages Starts

    Private Sub ShowSuccessMessage()

        MessageBox.Show("Grade Data successful removed!", "Success",
MessageBoxButtons.OK, MessageBoxIcon.Information)
    End Sub

    Private Sub ShowFailureMessage()

        MessageBox.Show("Failed to remove Grade Data unsuccessful!", "Failure",
MessageBoxButtons.OK, MessageBoxIcon.Information)
    End Sub

    'Response Messages Ends



    'Creating the CRUD METHODS - STARTS
```

```
'01. Get Method

Private Function GetData() As Data.DataTable

    command.CommandText = "SELECT DISTINCT

    sd.FirstName,

    sd.LastName,

    sd.StudentID,

    sag.TotalCreditHours,

    sag.SGPA,

    sag.GradeComment,

    gd1.Grade AS Grade1,

    gd2.Grade AS Grade2,

    gd3.Grade AS Grade3,

    gd4.Grade AS Grade4,

    gd5.Grade AS Grade5,

    gd6.Grade AS Grade6,

    gd7.Grade AS Grade7,


    cd1.CourseCode AS CourseCode1,

    cd2.CourseCode AS CourseCode2,

    cd3.CourseCode AS CourseCode3,

    cd4.CourseCode AS CourseCode4,

    cd5.CourseCode AS CourseCode5,

    cd6.CourseCode AS CourseCode6,
    cd7.CourseCode AS CourseCode7

    FROM tblStoreAllGrades sag

    JOIN tblStudentData sd ON gd1.StudID = sd.StudID

    JOIN tblGradeData gd1 ON sag.Grade1 = gd1.GradeID

    JOIN tblGradeData gd2 ON sag.Grade2 = gd2.GradeID

    JOIN tblGradeData gd3 ON sag.Grade3 = gd3.GradeID

    JOIN tblGradeData gd4 ON sag.Grade4 = gd4.GradeID

    JOIN tblGradeData gd5 ON sag.Grade5 = gd5.GradeID

    JOIN tblGradeData gd6 ON sag.Grade6 = gd6.GradeID

    JOIN tblGradeData gd7 ON sag.Grade7 = gd7.GradeID


    JOIN tblCourseData cd1 ON gd1.CourseID = cd1.CourseID

    JOIN tblCourseData cd2 ON gd2.CourseID = cd2.CourseID

    JOIN tblCourseData cd3 ON gd3.CourseID = cd3.CourseID

    JOIN tblCourseData cd4 ON gd4.CourseID = cd4.CourseID

    JOIN tblCourseData cd5 ON gd5.CourseID = cd5.CourseID

    JOIN tblCourseData cd6 ON gd6.CourseID = cd6.CourseID

    JOIN tblCourseData cd7 ON gd7.CourseID = cd7.CourseID

    LIMIT 7" '-- Added join condition for tblStudentData


    Dim adapter As New SQLiteDataAdapter(command)

    Dim dataTable As New Data.DataTable()

    adapter.Fill(dataTable)
```

```
        Return dataTable
    End Function
    'Auto Data Refresh Method - STARTS
    Private Sub RefreshDataGridView()
        dgvViewGradebook.DataSource = GetData()
        incrementedVariable()
        ' Set the initial columns you want at fixed display indices
        dgvViewGradebook.Columns("No").DisplayIndex = 0
        dgvViewGradebook.Columns("FirstName").DisplayIndex = 1
        dgvViewGradebook.Columns("LastName").DisplayIndex = 2
        dgvViewGradebook.Columns("StudentID").DisplayIndex = 3
        ' Set the course names as column headers and assign grades to specific courses
        For i As Integer = 1 To 7
            Dim columnName As String = "CourseCode" & i.ToString()
            Dim gradeColumnName As String = "Grade" & i.ToString()
            If dgvViewGradebook.Columns.Contains(columnName) Then
                dgvViewGradebook.Columns(columnName).HeaderText =
dgvViewGradebook.Rows(0).Cells(columnName).Value.ToString()
                dgvViewGradebook.Columns(columnName).DataPropertyName = gradeColumnName
                dgvViewGradebook.Columns(columnName).DisplayIndex = i + 3 ' Adjust display
index
            End If
        Next
        ' Set the indices for the remaining columns
        dgvViewGradebook.Columns("TCH").DisplayIndex = 18
        dgvViewGradebook.Columns("SGPA").DisplayIndex = 19
        dgvViewGradebook.Columns("Comment").DisplayIndex = 20
    End Sub
    'Auto Data Refresh Method - ENDS
    'Incremented Variable Starts
    Public Sub incrementedVariable()
        ' Get the number of rows in the DataGridView
        Dim rowCount As Integer = dgvViewGradebook.Rows.Count
        ' Loop through each row and assign incremented values to the "Number" column
        For i As Integer = 0 To rowCount - 1
            dgvViewGradebook.Rows(i).Cells("No").Value = i + 1
        Next
    End Sub
    'Search Button Logic Starts
    Public Sub ShowFilterResult(ProgramLevel As String, ProgramName As String, Year As
Integer, Semester As String)
        If cmbSemester.SelectedIndex = 1 Then
            lblSemester.Text = "First Semester" 'Show semester indicator
            Dim searchQuery As String = "SELECT DISTINCT
            sd.FirstName,
            sd.LastName,
            sd.StudentID,
            sag.TotalCreditHours,
            sag.SGPA,
            sag.GradeComment,
            gd1.Grade AS Grade1,
            gd2.Grade AS Grade2,
            gd3.Grade AS Grade3,
            gd4.Grade AS Grade4,
            gd5.Grade AS Grade5,
            gd6.Grade AS Grade6,
            gd7.Grade AS Grade7,

            cd1.CourseCode AS CourseCode1,
            cd2.CourseCode AS CourseCode2,
            cd3.CourseCode AS CourseCode3,
            cd4.CourseCode AS CourseCode4,
            cd5.CourseCode AS CourseCode5,
            cd6.CourseCode AS CourseCode6,
            cd7.CourseCode AS CourseCode7

        FROM tblStoreAllGrades sag
        JOIN tblStudentData sd ON gd1.StudID = sd.StudID
        JOIN tblGradeData gd1 ON sag.Grade1 = gd1.GradeID

        JOIN tblGradeData gd2 ON sag.Grade2 = gd2.GradeID

        JOIN tblGradeData gd3 ON sag.Grade3 = gd3.GradeID

        JOIN tblGradeData gd4 ON sag.Grade4 = gd4.GradeID

        JOIN tblGradeData gd5 ON sag.Grade5 = gd5.GradeID

        JOIN tblGradeData gd6 ON sag.Grade6 = gd6.GradeID

        JOIN tblGradeData gd7 ON sag.Grade7 = gd7.GradeID


        JOIN tblCourseData cd1 ON gd1.CourseID = cd1.CourseID

        JOIN tblCourseData cd2 ON gd2.CourseID = cd2.CourseID

        JOIN tblCourseData cd3 ON gd3.CourseID = cd3.CourseID

        JOIN tblCourseData cd4 ON gd4.CourseID = cd4.CourseID

        JOIN tblCourseData cd5 ON gd5.CourseID = cd5.CourseID

        JOIN tblCourseData cd6 ON gd6.CourseID = cd6.CourseID

        JOIN tblCourseData cd7 ON gd7.CourseID = cd7.CourseID

        LIMIT 7"

            Dim filterCmd As New SQLiteCommand(searchQuery, connection)

            ' Use parameterized query to avoid SQL injection

            filterCmd.Parameters.AddWithValue("@ProgramLevel", ProgramLevel)

            filterCmd.Parameters.AddWithValue("@ProgramName", ProgramName)

            filterCmd.Parameters.AddWithValue("@Year", Year)


            Dim resultAdapter As New SQLiteDataAdapter
```

```
resultAdapter.SelectCommand = filterCmd
' Create an instance of the DataTable class
Dim resultDataTable As New Data.DataTable()
resultAdapter.Fill(resultDataTable)
dgvViewGradebook.DataSource = resultDataTable
incrementedVariable()
' Set the initial columns you want at fixed display indices
dgvViewGradebook.Columns("No").DisplayIndex = 0
dgvViewGradebook.Columns("FirstName").DisplayIndex = 1
dgvViewGradebook.Columns("LastName").DisplayIndex = 2
dgvViewGradebook.Columns("StudentID").DisplayIndex = 3
' Set the course names as column headers and assign grades to specific courses
For i As Integer = 1 To 7
    Dim columnName As String = "CourseCode" & i.ToString()
    Dim gradeColumnName As String = "Grade" & i.ToString()
    If dgvViewGradebook.Columns.Contains(columnName) Then
        dgvViewGradebook.Columns(columnName).HeaderText =
dgvViewGradebook.Rows(0).Cells(columnName).Value.ToString()
    End If
    dgvViewGradebook.Columns(columnName).DataPropertyName = gradeColumnName
    dgvViewGradebook.Columns(columnName).DisplayIndex = i + 3 ' Adjust display
index
```

```
    Next
    ' Set the indices for the remaining columns
    dgvViewGradebook.Columns("TCH").DisplayIndex = 18
    dgvViewGradebook.Columns("SGPA").DisplayIndex = 19
    dgvViewGradebook.Columns("Comment").DisplayIndex = 20
ElseIf cmbSemester.SelectedIndex = 2 Then
    lblSemester.Text = "Second Semester" 'Show semester indicator
    Dim searchQuery As String = "SELECT DISTINCT
    sd.FirstName,
    sd.LastName,
    sd.StudentID,
    sag.TotalCreditHours,
    sag.SGPA,
    sag.GradeComment,
    gd8.Grade AS Grade8,
    gd9.Grade AS Grade9,
    gd10.Grade AS Grade10,
    gd11.Grade AS Grade11,
    gd12.Grade AS Grade12,
    gd13.Grade AS Grade13,
    gd14.Grade AS Grade14,

    cd8.CourseCode AS CourseCode8,
    cd9.CourseCode AS CourseCode9,

    cd10.CourseCode AS CourseCode10,

    cd11.CourseCode AS CourseCode11,

    cd12.CourseCode AS CourseCode12,

    cd13.CourseCode AS CourseCode13,

    cd14.CourseCode AS CourseCode14

FROM tblStoreAllGrades sag

JOIN tblStudentData sd ON gd8.StudID = sd.StudID

JOIN tblGradeData gd8 ON sag.Grade8 = gd8.GradeID

JOIN tblGradeData gd9 ON sag.Grade9 = gd9.GradeID

JOIN tblGradeData gd10 ON sag.Grade10 = gd10.GradeID

JOIN tblGradeData gd11 ON sag.Grade11 = gd11.GradeID

JOIN tblGradeData gd12 ON sag.Grade12 = gd12.GradeID

JOIN tblGradeData gd13 ON sag.Grade13 = gd13.GradeID

JOIN tblGradeData gd14 ON sag.Grade14 = gd14.GradeID


JOIN tblCourseData cd8 ON gd8.CourseID = cd8.CourseID

JOIN tblCourseData cd9 ON gd9.CourseID = cd9.CourseID

JOIN tblCourseData cd10 ON gd10.CourseID = cd10.CourseID

JOIN tblCourseData cd11 ON gd11.CourseID = cd11.CourseID

JOIN tblCourseData cd12 ON gd12.CourseID = cd12.CourseID

JOIN tblCourseData cd13 ON gd13.CourseID = cd13.CourseID

JOIN tblCourseData cd14 ON gd14.CourseID = cd14.CourseID
```

```
ORDER BY sag.AllGradesID DESC
LIMIT 7"
    Dim filterCmd As New SQLiteCommand(searchQuery, connection)
    ' Use parameterized query to avoid SQL injection
    filterCmd.Parameters.AddWithValue("@ProgramLevel", ProgramLevel)
    filterCmd.Parameters.AddWithValue("@ProgramName", ProgramName)
    filterCmd.Parameters.AddWithValue("@Year", Year)
    Dim resultAdapter As New SQLiteDataAdapter
    resultAdapter.SelectCommand = filterCmd
    ' Create an instance of the DataTable class
    Dim resultDataTable As New Data.DataTable()
    resultAdapter.Fill(resultDataTable)
    dgvViewGradebook.DataSource = resultDataTable
    incrementedVariable()
    ' Set the initial columns you want at fixed display indices
    dgvViewGradebook.Columns("No").DisplayIndex = 0
    dgvViewGradebook.Columns("FirstName").DisplayIndex = 1
    dgvViewGradebook.Columns("LastName").DisplayIndex = 2
    dgvViewGradebook.Columns("StudentID").DisplayIndex = 3
    ' Set the course names as column headers and assign grades to specific courses
    For i As Integer = 8 To 14
        Dim columnName As String = "CourseCode" & i.ToString()
        Dim gradeColumnName As String = "Grade" & i.ToString()
```

```
                If dgvViewGradebook.Columns.Contains(columnName) Then
                    dgvViewGradebook.Columns(columnName).HeaderText = dgvViewGradebook.Rows(0).Cells(columnName).Value.ToString()
                End If
                dgvViewGradebook.Columns(columnName).DataPropertyName = gradeColumnName
                dgvViewGradebook.Columns(columnName).DisplayIndex = i + 3 ' Adjust display index
            Next
            ' Set the indices for the remaining columns
            dgvViewGradebook.Columns("TCH").DisplayIndex = 18
            dgvViewGradebook.Columns("SGPA").DisplayIndex = 19
            dgvViewGradebook.Columns("Comment").DisplayIndex = 20
        Else
            ' Handle other cases (e.g., display an error message)
            MessageBox.Show("Invalid semester selection.")
        End If
    End Sub
    Private Sub btnFilterSearchF_Click(sender As Object, e As EventArgs) Handles btnFilterSearchF.Click
        Dim ProgramLevel As String = cmbSelectProgramLevel.SelectedItem
        Dim ProgramName As String = cmbSelectProgram.SelectedItem
        Dim Year As String = cmbSelectYear.SelectedItem
        Dim Semester As String = cmbSemester.SelectedItem
        'Change the top description to the selected Program and Year

        lblProgram.Text = ProgramLevel & " " & ProgramName & " " & Year

        ShowFilterResult(ProgramLevel, ProgramName, Year, Semester)

    End Sub

    Private Sub btnResetF_Click(sender As Object, e As EventArgs) Handles btnResetF.Click

        dgvViewGradebook.DataSource = GetData()

        incrementedVariable()

        ' Set the initial columns you want at fixed display indices

        dgvViewGradebook.Columns("No").DisplayIndex = 0

        dgvViewGradebook.Columns("FirstName").DisplayIndex = 1

        dgvViewGradebook.Columns("LastName").DisplayIndex = 2

        dgvViewGradebook.Columns("StudentID").DisplayIndex = 3

        ' Set the course names as column headers and assign grades to specific courses

        For i As Integer = 1 To 7

            Dim columnName As String = "CourseCode" & i.ToString()

           Dim gradeColumnName As String = "Grade" & i.ToString()

            If dgvViewGradebook.Rows.Count > 0 AndAlso
dgvViewGradebook.Columns.Contains(columnName) Then
                Dim cellValue As Object = dgvViewGradebook.Rows(0).Cells(columnName).Value

                If cellValue IsNot Nothing Then

                    dgvViewGradebook.Columns(columnName).HeaderText = cellValue.ToString()

                    dgvViewGradebook.Columns(columnName).DataPropertyName =
gradeColumnName
                    dgvViewGradebook.Columns(columnName).DisplayIndex = i + 3 ' Adjust display
index
```

```
            End If
         End If
      Next
      ' Set the indices for the remaining columns
      dgvViewGradebook.Columns("TCH").DisplayIndex = 18
      dgvViewGradebook.Columns("SGPA").DisplayIndex = 19
      dgvViewGradebook.Columns("Comment").DisplayIndex = 20
      '...
      cmbSelectProgramLevel.SelectedIndex = 0
      cmbSelectProgram.SelectedIndex = 0
      cmbSelectYear.SelectedIndex = 0
      lblSemester.Text = "First Semester"
   End Sub
   'Form Controls
   Private Sub btnBack_Click(sender As Object, e As EventArgs) Handles btnBack.Click
      frmDashboard.Show()
      Me.Close()
   End Sub
End Class
```

8. Update User/Administrator Account Logic

```
Imports System.Data.SQLite
Public Class frmUpdateUserProfile
```

```
'Database Connections
Private connection As SQLiteConnection
Private command As SQLiteCommand
Private getUserID As Integer
'Initializing SQLite Database connections
Public Sub New(userid As Integer)
    InitializeComponent()
    getUserID = userid
    Try
        connection = New SQLiteConnection("Data Source=gis.db;Version=3;Pooling=true;")
        connection.Open()
        command = connection.CreateCommand()
    Catch ex As Exception
        MessageBox.Show("Error connecting to the database: " & ex.Message, "Connection
Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
    End Try
End Sub
Private imagePath As String
Private Sub frmUpdateUserProfile_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
    LoadAdminData(getUserID)
End Sub
```

```
    'Response Messages Starts
    Private Sub ShowSuccessMessage()
        MessageBox.Show("User data updated successfully!", "Success", MessageBoxButtons.OK,
MessageBoxIcon.Information)
    End Sub
    Private Sub ShowFailureMessage()
        MessageBox.Show("Fail to update user data!", "Failure", MessageBoxButtons.OK,
MessageBoxIcon.Information)
    End Sub
    'Response Messages Ends
    'Creating the CRUD METHODS - STARTS
    '01. Get Method
    Private Sub LoadAdminData(getUserID As Integer)
        Dim query As String = "SELECT * FROM tblUserData WHERE UserID = @getUserID"
        Dim command As New SQLiteCommand(query, connection)
        command.Parameters.AddWithValue("@getUserID", getUserID)
        Dim reader As SQLiteDataReader = command.ExecuteReader()
        While reader.Read()
                Dim value1 As String = reader.GetString(1)
                Dim value2 As String = reader.GetString(2)
                Dim value3 As String = reader.GetString(3)
                Dim value4 As String = reader.GetString(4)
                Dim value5 As String = reader.GetString(5)
```

```
            Dim value7 As String = Convert.ToString(reader.GetValue(7))
            txtFName.Text = value1
            txtLName.Text = value2
            txtSecretRestorePin.Text = value3
            txtUsername.Text = value4
            txtPassword.Text = value5
            Dim ImagePath As String = value7
            If Not String.IsNullOrEmpty(ImagePath) AndAlso IO.File.Exists(ImagePath) Then
                pctUserImage.Image = Image.FromFile(ImagePath)
            Else
                ' Handle the case where the image path is empty or invalid
                pctUserImage.Image = Nothing
            End If
        End While
        reader.Close()
    End Sub
    '02. Update Method
    Private Sub UpdateData(UserID As Integer, firstname As String, lastname As String, secretpin
As String, username As String, password As String, userrole As String, imagePath As String)
        command.CommandText = "UPDATE tblUserData SET FirstName = @FirstName,
LastName = @LastName, SecretPin = @SecretPin, Username = @Username, Password =
@Password,
        UserRole = @UserRole, ImagePath = @ImagePath WHERE UserID = @UserID"
        command.Parameters.AddWithValue("@UserID", UserID)
        command.Parameters.AddWithValue("@FirstName", firstname)
        command.Parameters.AddWithValue("@LastName", lastname)
        command.Parameters.AddWithValue("@SecretPin", secretpin)
        command.Parameters.AddWithValue("@Username", username)
        command.Parameters.AddWithValue("@Password", password)
        command.Parameters.AddWithValue("@UserRole", userrole)
        command.Parameters.AddWithValue("@ImagePath", imagePath)
        If (command.ExecuteNonQuery() > 0) Then
            ShowSuccessMessage()
        Else
            ShowFailureMessage()
        End If
    End Sub
    'Clear Data Method - STARTS
    Private Sub ClearData()
        txtFName.Text = " "
        txtLName.Text = " "
        txtSecretRestorePin.Text = ""
        txtUsername.Text = ""
        txtPassword.Text = ""
        txtConfirmPassword.Text = ""
```

```
        pctUserImage.Image = Nothing

    End Sub

    'Clear Data Method - ENDS

    Private Sub btnSelectImage_Click(sender As Object, e As EventArgs) Handles
btnSelectImage.Click

        Dim openFileDialog As New OpenFileDialog()

        openFileDialog.Filter = "Image Files (*.jpg, *.png, *.bmp)|*.jpg;*.png;*.bmp|All Files (*.*)|*.*"
        openFileDialog.InitialDirectory =
Environment.GetFolderPath(Environment.SpecialFolder.MyPictures)
        If openFileDialog.ShowDialog() = DialogResult.OK Then

            pctUserImage.Image = Image.FromFile(openFileDialog.FileName)

            ' Store the image path for later use when saving to the database

            imagePath = openFileDialog.FileName

        End If

    End Sub

    'Handling User Interactions

    Private Sub btnUpdateAccount_Click(sender As Object, e As EventArgs) Handles
btnUpdateAccount.Click

        Dim UserID As Integer = getUserID

        Dim firstname As String = txtFName.Text

        Dim lastname As String = txtLName.Text

        Dim secretpin As String = txtSecretRestorePin.Text

        Dim username As String = txtUsername.Text

        Dim password As String = txtPassword.Text

        Dim confirmpassword As String = txtConfirmPassword.Text
        Dim userrole As String = "Admin"

        If (confirmpassword = password) Then

            UpdateData(UserID, firstname, lastname, secretpin, username, password, userrole,
imagePath)

            ClearData()

        Else

            ' Invalid credentials

            MessageBox.Show("Passwords do not match.")

        End If

    End Sub

    'Form Controls

    Private Sub btnBack_Click(sender As Object, e As EventArgs) Handles btnBack.Click

        frmDashboard.Show()

        Me.Close()

    End Sub

    Private Sub frmUpdateUserProfile_FormClosed(sender As Object, e As
FormClosedEventArgs) Handles MyBase.FormClosed
        Me.Dispose() ' Dispose the form to ensure it is recreated when reopened
    End Sub
End Class
```