Blood Donation Management System

Session: 2022 – 2026

Submitted by:

Muhammad Wali Ahmad 2022-CS-65

Supervised by:

Maida Shahid

Department of Computer Science

University of Engineering and Technology

Lahore Pakistan

Table of Contents

1. INTRODUCTION:

1.1 Overview:

Problem Statement: During emergency situations or surgeries, finding the required blood group quickly and efficiently can be a challenging task. This creates a need for a blood donation management system that simplifies the process and provides a platform for individuals to search for their desired blood group.

Solution: The blood donation management system aims to address the problem by utilizing technology to streamline the blood donation process. It provides an organized system that connects donors and individuals in need of specific blood types, making it easier to locate and acquire the required blood group in a timely manner.

Significance: The development and implementation of a blood donation management system can have a positive impact on both computer science and healthcare. By leveraging technology, the system improves the efficiency and accessibility of blood donation processes, ensuring the availability of safe and adequate blood supplies. This has the potential to significantly improve healthcare outcomes and save lives, making it an essential tool for patients, healthcare providers, and blood donors.

1.2 Objectives:

Enable Easy Blood Group Search: The primary objective of the blood donation management system is to provide a user-friendly platform where individuals can quickly and easily search for their desired blood group during emergency situations or surgeries.

Streamline Blood Donation Processes: The system aims to automate and streamline various aspects of the blood donation process, including managing donor information, maintaining records of donated blood units, and organizing blood drives or collection centers. This objective ensures efficient management of blood resources.

Improve Healthcare Outcomes: By ensuring the availability of safe and adequate blood supplies for transfusions, the system aims to contribute to better healthcare outcomes. This objective involves reducing the time and effort required to find matching blood donors, ultimately leading to improved patient care and potentially saving lives.

1.3 Intended Functionality:

User-Friendly Interface: The blood donation management system will have a user-friendly interface that allows individuals to easily search for specific blood groups based on their requirements. The interface will be intuitive, responsive, and accessible to a wide range of users.

Real-Time Availability: The system will provide real-time information on the availability of different blood groups. This feature enables quick identification and location of the required blood type, ensuring timely access to blood donors during emergency situations or surgeries.

Donor and Donation Management: The system will facilitate the management of donor information, including registration, eligibility screening, and contact details. It will also maintain records of donated blood units, ensuring accurate tracking and inventory management.

Blood Drive Organization: The system will enable the organization and coordination of blood drives and collection centers. This functionality includes scheduling, volunteer management, and communication tools to optimize the efficiency of blood donation campaigns.

Data Security and Privacy: The blood donation management system will prioritize the security and privacy of donor and patient information. It will incorporate robust security measures and adhere to data protection regulations to ensure the confidentiality and integrity of sensitive data.

2. OOP-CONCEPTS:

In the context of the blood donation management system, object-oriented programming (OOP) principles and concepts can be applied to enhance the design, organization, and functionality of the system. Here are some examples of how OOP can be associated with different aspects of the system:

2.2 Association:

Association is of two types aggregation and composition both are applied in the management system. Aggregation is present between different classes like all classes of UI to classes of BL and DL. Also, there is aggregation between Employee class with Person class employee contain

list of persons. While composition is present between DL classes to the respective BL classes like PersonCRUD.cs has composition with Person.cs.

2.2 Encapsulation:

Encapsulation is a fundamental principle of OOP that involves bundling data and related methods within a class, hiding the internal implementation details from other parts of the system. In the blood donation management system, encapsulation can be employed to encapsulate the properties and methods of classes such as Donor, Recipient and Employee protecting their internal state and providing controlled access to their data.

2.3 Inheritance:

Inheritance is another key concept in OOP that allows the creation of hierarchical relationships between classes. In the blood donation management system, inheritance can be utilized to establish relationships between different types of donors or recipients to a single person class. For example, a Donor class could inherit from a more general Person class, inheriting its properties and methods while adding specific characteristics or behaviors. Also, there is inheritance between Users of the systems.

2.4 Polymorphism:

Polymorphism enables objects of different classes to be treated interchangeably, allowing for flexibility and extensibility in the system. In the blood donation management system, polymorphism can be applied when handling different types of donors and recipients. Methods or functions that operate on donors or recipients can be designed to accept objects of the general class type named Person, enabling the system to handle various specific instances transparently. It also helpful in when storing into list.

3. COMPARE WITH PROCEDURAL PROGRAMING:

Following are the different aspects in which OOP is better than procedural programing:

3.1 Reusability and Code Sharing:

OOP promotes code reusability through the concept of inheritance. Inheritance allows classes to inherit properties and behaviors from other classes, reducing the need to rewrite code. This

not only saves development time but also makes the codebase more efficient and easier to maintain. In procedural programming, reusing code requires copying and pasting or creating separate functions, which can lead to code redundancy and maintenance issues.

3.2 Maintainability and Scalability:

OOP promotes code maintainability and scalability. With encapsulation and modularity, making changes to a specific functionality or fixing issues becomes easier because the affected code is localized within the relevant class. Additionally, OOP's ability to extend existing classes through inheritance allows for the addition of new features without modifying the existing codebase extensively. In procedural programming, making changes or adding features often involves modifying multiple functions, increasing the likelihood of introducing errors and making maintenance more challenging.

3.3 Code Readability and Understandability:

OOP encourages a more natural representation of real-world entities and their relationships. This makes the codebase more readable and understandable, as classes and objects closely resemble the entities and interactions they model. Procedural programming, on the other hand, can lack this intuitive representation, making it harder to grasp the overall system design and the relationships between different parts of the code.

3.4 Data Security and Abstraction:

OOP allows for the implementation of data security through the concept of encapsulation. By hiding the internal implementation details of a class, OOP ensures that data can only be accessed and modified through designated methods, reducing the risk of unintended modifications or data corruption. Procedural programming typically lacks built-in mechanisms for data security and abstraction, making it more prone to data integrity issues and unauthorized access.

4. DESIGN PATTERN IMPLEMENTATION:

In the blood donation management system, the utilization of design patterns helps ensure modularity, separation of concerns, and maintainability. Here's how the project incorporates the Business Logic (BL), Data Access Layer (DL), and User Interface (UI) design patterns:

4.1 Business Logic (BL) Design Pattern:

The BL design pattern focuses on encapsulating the business rules and logic of the application. It ensures that the core functionality and operations of the system are independent of the underlying data storage or user interface.

In the blood donation management system: The BL design pattern can be implemented using the **Model-View-Controller** (MVC) architectural pattern. The model represents the business logic and operations of the system. It encapsulates functionalities such as single employee functionalities and his credentials, donor or recipient details, and their related functionality. The controllers handle the interaction between the user interface and the business logic. They receive user input, invoke appropriate operations in the model, and update the views accordingly. The views represent the user interface components, which display information to the users and receive their input. They communicate with the controllers to trigger actions and display updated information. By implementing the BL design pattern, the blood donation management system achieves a separation of concerns between the business logic and the user interface, making it easier to maintain and modify each component independently.

4.2 Data Access Layer (DL) Design Pattern:

The DL design pattern focuses on managing the interaction between the application and the underlying data storage or database. It abstracts the data access operations, ensuring that the business logic is decoupled from the specifics of the data storage implementation.

In the blood donation management system: The DL design pattern can be implemented using the **Repository pattern**. The repositories provide an abstraction layer between the business logic and the database. They handle data retrieval, storage, and update operations. The repositories encapsulate the details of the data access, such as querying the data, mapping data to objects, and ensuring data integrity and consistency. By applying the DL design pattern, the blood donation management system achieves separation between the business logic and the data storage implementation. This separation allows for easier maintenance, scalability, and the potential to switch or upgrade the underlying database without impacting the overall functionality of the system.

4.3 User Interface (UI) Design Pattern:

The UI design pattern focuses on structuring and organizing the user interface components to ensure a consistent and user-friendly experience. It separates the presentation layer from the underlying business logic and data access operations.

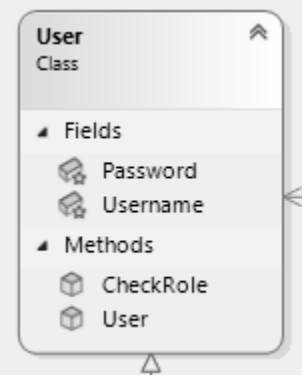
In the blood donation management system: The UI design pattern can be implemented using the **Model-View-ViewModel** (MVVM) architectural pattern. The model represents the data and business logic, while the view model acts as an intermediary between the model and the view. The view model exposes properties and commands that the view can bind to, facilitating data binding and handling user actions. The view represents the visual components of the user interface, displaying the data from the view model and capturing user input. By utilizing the UI design pattern, the blood donation management system separates the UI concerns from the underlying business logic and data access operations. This separation allows for easier modification, testing, and customization of the user interface, without impacting the core functionality of the system.

5. CLASSES DETAILS:

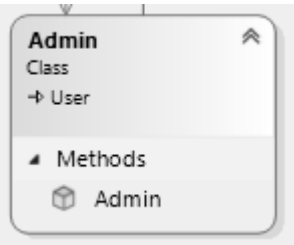
Explanations of classes in the blood donation management system along with class diagram to illustrate their responsibilities:

5.1 Business Logic (BL) Classses:

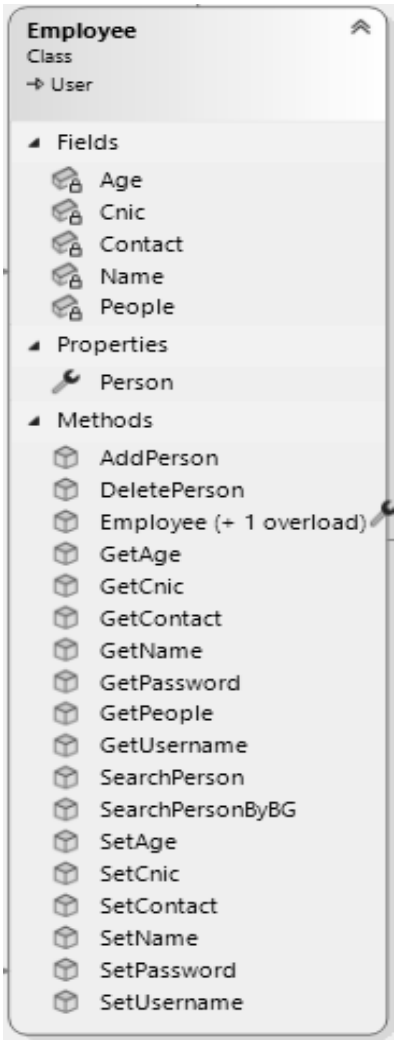
- **User:**
  This is the main class for login to the application it is the parent class for three users i.e., admin, employee and guest.



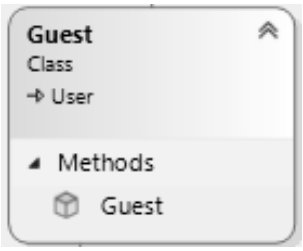
- **Admin:**
  Admin is the child class of user and it is responsible for employees operation.



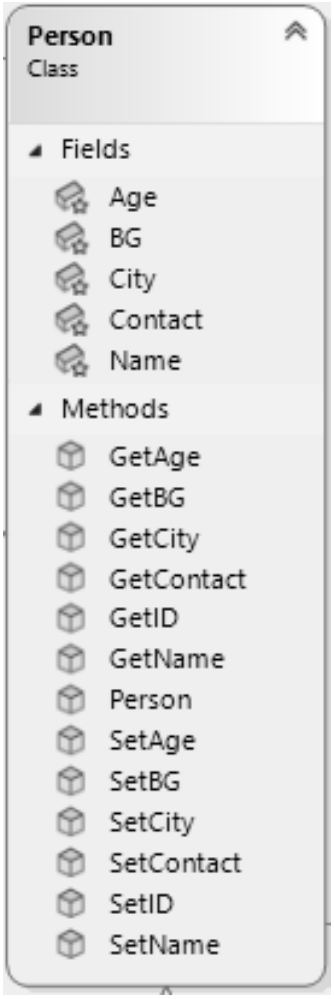
- **Employee:**
  Employee is the child class of user and it is responsible for donors and recipient operation.



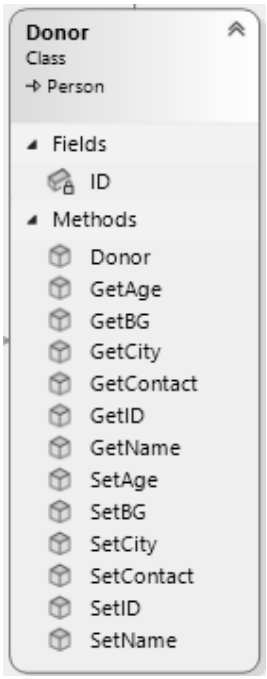
- **Guest:**
  Guest is the child class of user and it is responsible add request for blood group.



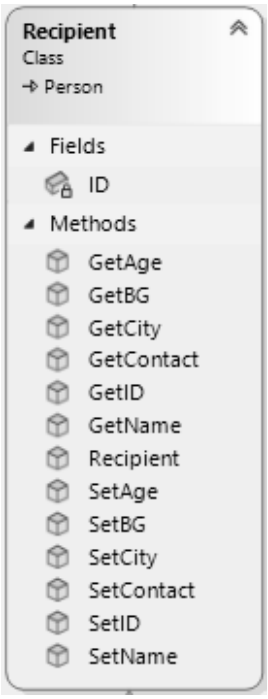
- **Person:**
  Person is also a parent class for donor and recipient. There is dynamic polymorphism present in this class with his child classes.



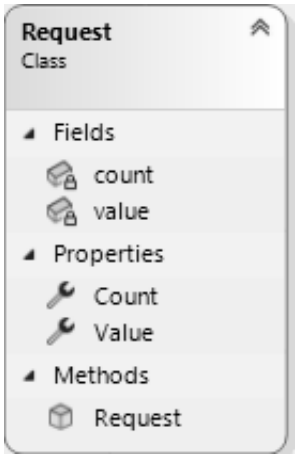
- **Donor:**
  It is the child class of person and it contains respective attributes of the donor.



- **Recipient:**
  It is the child class of person and it contains respective attributes of the recipient.

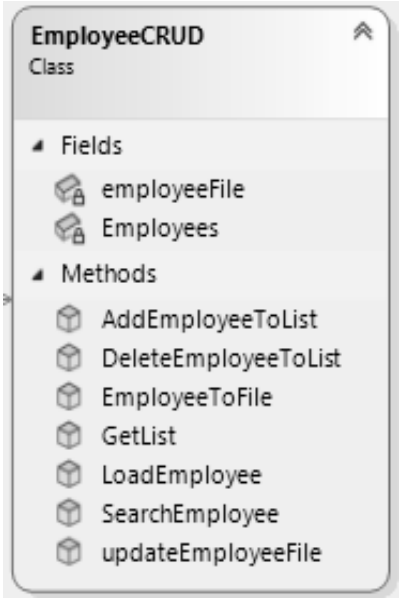


- **Request:**
  It is helping class to count the requests for respective blood group.

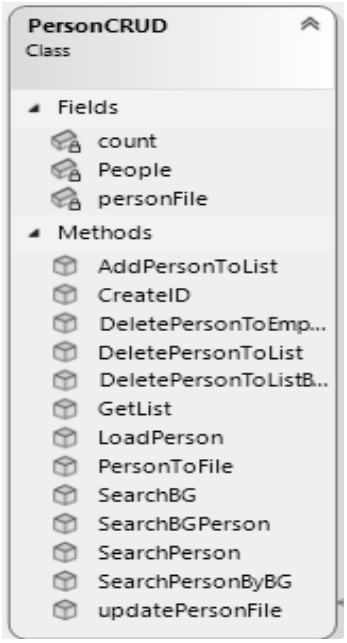


5.2 Data Access Layer (DL) Classses:

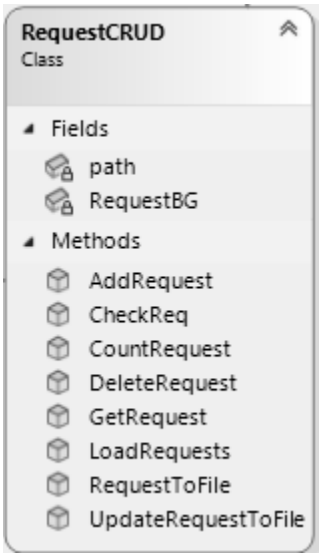
- **EmployeeCRUD:**
  This class is responsible for data handling of employees it stores data into file when changes held or add new employees and also load data when application start.



- **PersonCRUD:**
  This class is responsible for data handling of person i.e., donors and recipients it stores data into file when changes held or add new person and also load data when application start.

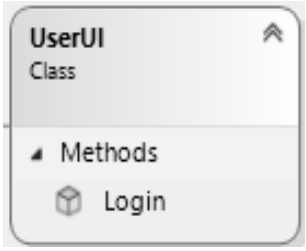


- **RequestCRUD:**
  This class is responsible for data handling of request made by guest user it stores request into file when request fulfill it deletes it or add new request when made and also load data when application start.

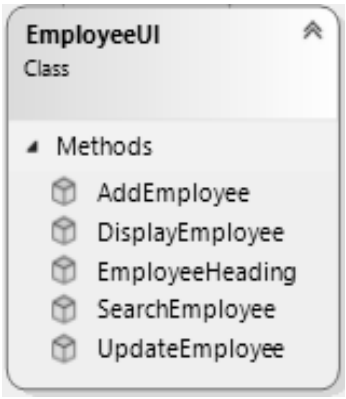


5.3 User Interface (UI) Classses:

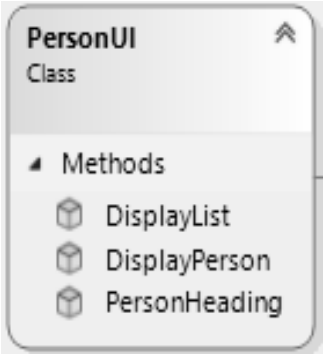
- **UserUI:**
  This class provide user interface for login to the application.



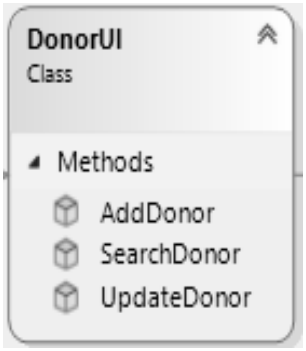
- **EmployeeUI:**
  This class provide interface for the employee operations that is adding new employee and different others operations.



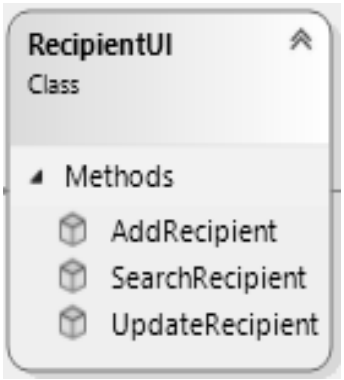
- **PersonUI:**
  This class provide interface for the person operations that is show all persons and heading to show on the top of all persons.



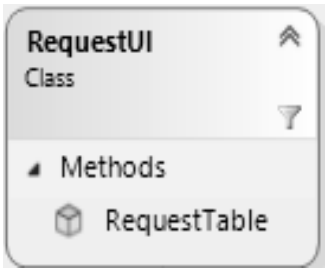
- **DonorUI:**
  This class provide interface for the donor operations that is adding new donors and different others operations.



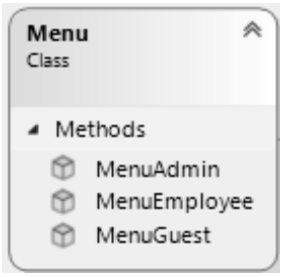
- **RecipientUI:**
  This class provide interface for the recipient operations that is adding new recipient and different others operations.



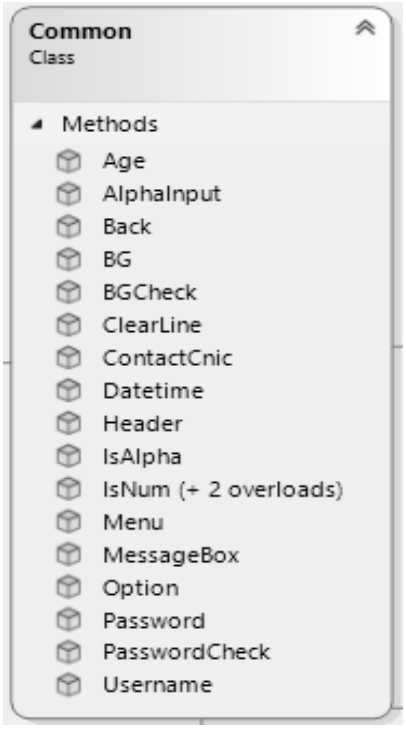
- **RequestUI:**
  This class provide interface for the requests operations that is adding new requests and show all requests.



- **Menu:**
  This class contains all menus for the application.



- **Common:**
  This class contains all common functions for the application. Like header, validations functions etc.



6. CONCLUSION:

In the conclusion I will summarize my project idea and its achievements, challenged faced during developments and the new things I learnt from this project.

6.1 Summarization and Achievements:

The blood donation management system is a technology-driven solution designed to address the challenges associated with finding specific blood groups during emergency situations or surgeries. By leveraging the principles of object-oriented programming, the system improves the efficiency and accessibility of the blood donation process, ultimately contributing to better healthcare outcomes.

Throughout the development of the project, several achievements have been made. The system provides a user-friendly interface that allows individuals to easily search for their desired blood group in a short time. Real-time availability of blood group information ensures timely access to suitable blood donors, potentially saving lives. The system also incorporates features such as donor and donation management, blood drive organization, and data security measures, enhancing the overall functionality and usability of the system.

6.2 Challenges:

However, the project also encountered some challenges along the way. One significant challenge was the communication between different classes of the system. Managing a large data of donors and recipients along with their blood groups and also respective employees. while maintaining data maintainability and validation also posted its own set of challenges. Additionally, the most challenging moment for me is ORM during file-handling.

6.2 Lesson Learned:

Throughout the project, several valuable lessons were learned. It became evident that effective planning and requirement gathering are crucial for the successful development of such a complex system. Modularity and separation of concerns played a vital role in achieving maintainability and flexibility. The implementation of design patterns, such as the BL, DL, and UI patterns, helped in achieving a structured and organized codebase. Furthermore, continuous testing and quality assurance were imperative to identify and resolve any issues early on.

Student Reg. No.: 2022-CS-65 **Student Name:** Muhammad Wali Ahmad

	A-Extensive Evidence	**B-Convincing Evidence**	**C-Limited Evidence**	**D-No Evidence**
Documentation Formatting **Grade:**	All the documentation meets all the criteria.	Documentation is well formatted but some of the criteria is not fulfilled.	Documentation is required a lot of improvement.	Documentation is not Available
Documentation Formatting Criteria: In **Binder**, **Title** Page, **Header**-Footers, Font **Style**, Font **Size** all are all consistence and according to given **guidelines**. Project **Poster** is professionally design and well presented				
Documentation Contents **Grade:**	Documentation includes all of the criteria.	Documentation meet more than 80% of the criteria given.	Documentation meet more than 50% of the criteria.	When the documentation meet less than 50% of the criteria.
Documentation Contents Criteria: Title Page - **Table** of Contents - Project **Abstract** - **Functional** Requirements - **Wire** Frames –**Data Flow** Diagram- **Data** Structure (Arrays)-**Function** Headers and Description -Project **Code.** - **Weakness** in the Project and **Future** Directions. - **Conclusion** and What your **Learn** from the Project and Course and What is your **Future** Planning.				
Project Complexity **Grade:**	Project has at least 2 user's types and each user has at least 5 functionalities.	Project complexity meet 80% criteria given in extensive evidence	Project complexity meet 50% criteria given in extensive evidence	Project complexity meet less than 50% criteria given in extensive evidence
Code Style **Grade:**	All Code style criteria is followed	All code style criteria followed but some improvements required	lot of improvements required in coding style.	**Did not follow** code style,
Code Style Criteria: Consistent code style. Code is well indented. Variable and Function names are well defined. White Spaces are well used. Comments are added.				
Code Documentation Mapping **Grade:**	Code and documentation is synchronized.	Code and documentation does not synchronized at **some** places	Code and documentation does not synchronized at **many** places	Code and documentation **does not** synchronized.
Data Structure (Arrays) **Grade:**	Data structure is sufficient for the project requirements	Data Structure is sufficient but require improvement to meet project requirements.	Data structure is not sufficient and need a lot of improvement	Data Structure is not properly identified and declared.
Modularity **Grade:**	Meet all Modularity criteria	Meet all Modularity criteria but at some places it is missing	Do not sufficiently meet the modularity criteria.	No modularity or very minimum modularity.
Modularity criteria: Functions are defined for each major feature. Functions are independent (identify from parameter list and return types).				
Validations **Grade:**	Validations on all number type inputs are applied	Validations are applied but at some places it is missing.	Validations are missing at lot of places	No Validations are used
File Handling **Grade:**	Separate files for separate data. Data in csv format	File handing require some improvements	File handing require a lot of improvements	Not implemented
Aesthetics of the User Interface **Grade:**	UI is presentable. Proper coloring, Headers and clear screen is done	UI require some improvements	UI require a lot of improvements	Not implemented
Presentation and Demo **Grade:**	Presentation and Demo was 100% working	Presentation and Demo require some improvements	Presentation and Demo require a lot of improvements	Presentation was not ok and Demo was not working
Student Understanding with the Code. **Grade:**	Student has complete understanding how the code is working and knows the concept.	Student has good understand but some place he does not know the concepts	Student has a very little understand and lack the major concepts.	Student does not have any level of understanding of the code.

Checked by:	
Comments:	

Student Reg. No.: 2022-CS-65 **Student Name:** Muhammad Wali Ahmad

	A-Extensive Evidence	**B-Convincing Evidence**	**C-Limited Evidence**	**D-No Evidence**
Documentation Formatting **Grade:**	All the documentation meets all the criteria.	Documentation is well formatted but some of the criteria is not fulfilled.	Documentation is required a lot of improvement.	Documentation is not Available
Documentation Formatting Criteria: In **Binder**, **Title** Page, **Header**-Footers, Font **Style**, Font **Size** all are all consistence and according to given **guidelines**. Project **Poster** is professionally design and well presented				
Documentation Contents **Grade:**	Documentation includes all of the criteria.	Documentation meet more than 80% of the criteria given.	Documentation meet more than 50% of the criteria.	When the documentation meet less than 50% of the criteria.
Documentation Contents Criteria: Title Page - **Table** of Contents - Project **Abstract** - **Functional** Requirements - **Wire** Frames –**Data Flow** Diagram- **Data** Structure (Arrays)-**Function** Headers and Description -Project **Code.** - **Weakness** in the Project and **Future** Directions. - **Conclusion** and What your **Learn** from the Project and Course and What is your **Future** Planning.				
Project Complexity **Grade:**	Project has at least 2 user's types and each user has at least 5 functionalities.	Project complexity meet 80% criteria given in extensive evidence	Project complexity meet 50% criteria given in extensive evidence	Project complexity meet less than 50% criteria given in extensive evidence
Code Style **Grade:**	All Code style criteria is followed	All code style criteria followed but some improvements required	lot of improvements required in coding style.	**Did not follow** code style,
Code Style Criteria: Consistent code style. Code is well indented. Variable and Function names are well defined. White Spaces are well used. Comments are added.				
Code Documentation Mapping **Grade:**	Code and documentation is synchronized.	Code and documentation does not synchronized at **some** places	Code and documentation does not synchronized at **many** places	Code and documentation **does not** synchronized.
Data Structure (Arrays) **Grade:**	Data structure is sufficient for the project requirements	Data Structure is sufficient but require improvement to meet project requirements.	Data structure is not sufficient and need a lot of improvement	Data Structure is not properly identified and declared.
Modularity **Grade:**	Meet all Modularity criteria	Meet all Modularity criteria but at some places it is missing	Do not sufficiently meet the modularity criteria.	No modularity or very minimum modularity.
Modularity criteria: Functions are defined for each major feature. Functions are independent (identify from parameter list and return types).				
Validations **Grade:**	Validations on all number type inputs are applied	Validations are applied but at some places it is missing.	Validations are missing at lot of places	No Validations are used
File Handling **Grade:**	Separate files for separate data. Data in csv format	File handing require some improvements	File handing require a lot of improvements	Not implemented
Aesthetics of the User Interface **Grade:**	UI is presentable. Proper coloring, Headers and clear screen is done	UI require some improvements	UI require a lot of improvements	Not implemented
Presentation and Demo **Grade:**	Presentation and Demo was 100% working	Presentation and Demo require some improvements	Presentation and Demo require a lot of improvements	Presentation was not ok and Demo was not working
Student Understanding with the Code. **Grade:**	Student has complete understanding how the code is working and knows the concept.	Student has good understand but some place he does not know the concepts	Student has a very little understand and lack the major concepts.	Student does not have any level of understanding of the code.

Checked by:	
Comment:	