**Student Name: Weight: 40%**

**Student ID:** **Marks:** **/100**

# Project:

## Scenario

Choose one of the following topics and create a **MAUI** **desktop** application connected to the database:

1. Learning Management System
2. Restaurant Management System
3. Salaries Management System
4. Library Management System
5. Gym Management System
6. Hospital Management System
7. House Utilities Management System
8. Mechanic shop Management System
9. Employee Management System
10. Events Management System

## Equipment and Materials

For this assignment, you will need:

* Visual Studio IDE

## Instructions

1. After choosing one of the topics above, create a UML class diagram, ERD (Entity Relationship Diagram), and UI mock-up to plan your project and write the program of the system you chose.
2. Working outside of class time, complete the submission sections of this assignment.
3. Review the grading criteria for the group submissions.
4. See the course schedule and/or Brightspace for due dates.

**Submission**

1. Create the code for a program that meets the requirements described bellow.
2. Test your code against the expected requirements.
3. Check your program against the detailed marking criteria at the end of this document.
4. Submit the following to Brightspace as a group (*Only one copy is required per group, and any of the group members may submit*):

* Readme file to explain the project functionality and problem that it solves
* Project code
* Presentation video of the project showing how it runs
* GitHub URL for your program code
* A copy of the test output (.txt file)/screenshots of the project

**Peer Assessment**

Each student must also complete a peer assessment of their group members. Your instructor will provide further submission details.

**Presentation**

In your presentation, you should include the following:

1. Introduction:
   * Briefly introduce the project and its purpose.
   * Mention the technologies used in the project such as classes, interfaces, abstract classes, MAUI GUI and exceptions.
   * Provide an overview of the project's goals and functionality.
2. Functionality:
   * Describe the core functionality of the project.
   * Detail how classes, interfaces, and abstract classes are used to achieve this functionality.
3. Classes and Inheritance:
   * Define the classes that are used in the project.
   * Explain how inheritance is used to create a hierarchy of classes.
   * Discuss how interfaces/abstract classes are implemented in the project.
4. MAUI GUI:
   * Explain how MAUI GUI was used to create the project's user interface.
5. Exceptions:
   * Explain how exceptions are handled in the project.
   * Provide examples of how exceptions are used in the project.
6. Conclusion:
   * Summarize the main points of the rubric.
   * Reiterate the project's goals and functionality.
   * Provide any final thoughts or considerations.

## Marking Criteria

### Individual Submission

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Not submitted (0%)** | **Submitted (100%)** | **Marks** |
| **Peer assessment** | * Not submitted | * Completed for the group member | **/5** |

### Group Submission

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Needs Improvement (0–50%)** | **Good (51–75%)** | **Excellent (76–100%)** | **Marks** |
| **Working Code** | * The project doesn’t run in all scenarios * Syntax errors | * The project runs in all scenarios * Some Syntax errors | * The project runs in all scenarios * No Syntax errors | **/10** |
| **Validation** | * The application does not perform any input validation, which creates significant security risks, data corruption, or crashes. | * The application performs some input validation to ensure that some user input is safe, secure, and error-free, with limited error messages and feedback to the user. | * The application performs excellent input validation to ensure that all user input is safe, secure, and error-free, with proper error messages and feedback to the user. | **/10** |
| **Design** | * The application has a poor user interface that is not visually appealing, difficult to navigate, and provides an unsatisfactory user experience. | * The application has a good user interface that is mostly visually appealing, easy to navigate, and provides a satisfactory user experience. | * The application has a well-designed user interface that is visually appealing, easy to navigate, and provides an enjoyable user experience. | **/15** |
| **Pages** | * The application has only one page/screen that is poorly designed, not fully functional, and not responsive, with confusing navigation and no feedback to the user. | * The application has two or three pages/screens that are mostly well-designed, functional, and responsive, with some navigation and feedback to the user. | * The application has three or more pages/screens that are well-designed, functional, and responsive, with proper navigation and feedback to the user. | **/10** |
| **Use of Interfaces and Abstract Classes** | * The application does not use interfaces and abstract classes, which severely limits the separation of concerns, flexibility and modularity, and maintainability and testability of the code. | * The application makes little use of interfaces and abstract classes, which hinders the separation of concerns, flexibility and modularity, and maintainability and testability of the code. | * The application makes extensive use of interfaces and abstract classes to provide a clear separation of concerns, enable flexibility and modularity, and enhance maintainability and testability of the code. | **/10** |
| **Use of Exceptions** | * The application does not use exceptions to handle errors and unexpected events, which leads to crashes, data corruption, or security vulnerabilities. | * The application makes some use of exceptions to handle some errors and unexpected events, providing limited error messages, and ensuring partial handling of resources and data integrity. | * The application makes extensive use of exceptions to handle errors and unexpected events, providing meaningful error messages, and ensuring proper handling of resources and data integrity. | **/10** |
| **Style** | * Indentation – not consistent * Readability – poor variable names * Documentation: * No comments are included at the top. * No comments indicating major code sections or what they do | * Indentation – some parts are consistent and some are not * Readability – some variable names are not ideal * Documentation: * Comments at the top are missing or incomplete. * Comments indicating major code sections and what they do are incomplete | * Indentation – consistent * Readability – good variable names * Documentation: * Comments at the top are complete and include name, date, program description including details on inputs, processing and outputs  (4–5 sentences minimum). * Comments indicate major code sections and what they do | **/10** |
| **Connection to database** | * The application does not use database integration, which results in incomplete or inaccurate data, poor performance, or security vulnerabilities. | * The application demonstrates little use of database integration, including flawed data modeling, inefficient data access, and poor error handling, while violating coding practices | * The application demonstrates excellent use of database integration, including proper data modeling, efficient data access, and robust error handling, while adhering to good coding practices | **/10** |
| **Total** | | | | **/100** |