BICT IT8118: Advanced Programming

Group Project Instructions

Contents

Project Description

Project Overview:

Project Objective:

In this project, you will develop an **Equipment Rental Management System** that enables rental companies to manage their inventory, process rental requests, and track rented equipment. The system will allow customers to browse available equipment, request rentals, track their orders, and return items. It will include features such as authentication and authorization, inventory management, rental request processing, return tracking, notifications, dashboards, and more.
The system will have a **Windows Forms application** and an **ASP.NET Core MVC application**, both of which will use a shared database and **Entity Framework Core models**.

IDE: Visual Studio 2022 **Language**: Visual C# (.NET Core) **.NET:** 6 (or newer)

Business Scenario:

A company that rents out equipment (e.g., power tools, cameras, event supplies, construction equipment, etc.) wants to transition to an online system to streamline its operations. The system will help the company manage its rental inventory, accept rental requests from customers, track active rentals, and handle returns efficiently.

Company Staff Will Be Able To:

- Manage rental inventory (add, update, delete equipment with details like name, category, rental price, availability status).
- Process rental requests from customers, approving or rejecting them based on availability.
- Assign and track equipment once a rental is approved.
- Monitor rental durations and overdue items.
- Track equipment returns and update availability status.
- View and generate reports about customers or rentals history.

Customers Will Be Able To:

- Browse available equipment by category and search based on availability.
- Submit rental requests with preferred rental dates.
- Receive notifications for their rental requests.
- Track active rentals and return deadlines.
- Return rented items and leave feedback.

(**Note:** manage usually includes functionality to Create, Read, Update, and Delete).

Project Groups:

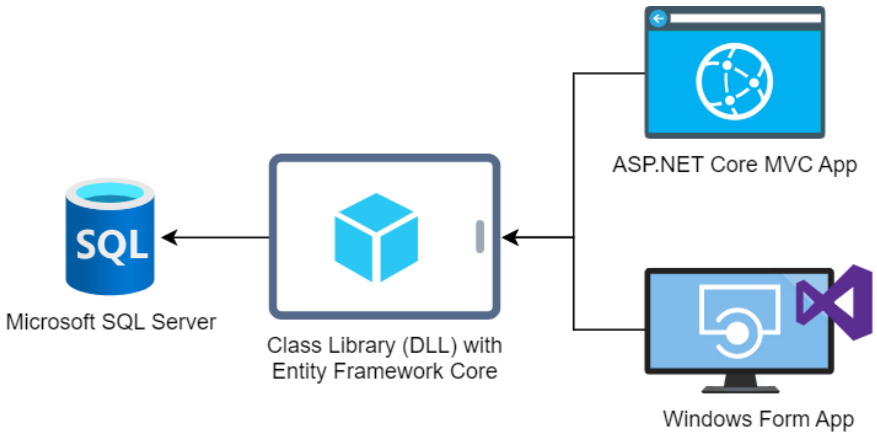
Group members assignment will be coordinated in class.

- The required number of students in each project group is 4-5 as far as possible. Each member of a project group should take a leadership role on one of the project components. However, all members of the group should work together on all the components.
- The project is assessed as a group project.

Project Components:

The project will have the following main components:

- Database (using Microsoft SQL Server)
- Class Library Project (using Entity Framework Core)
- ASP.NET Core MVC Project
- Windows Forms Application Project



Project Tasks:

The project is split into tasks to allow easier allocation between the group members, where each member can take the lead in a particular task. However, the success or failure at each task will affect the whole project. Thus, it is recommended to have group discussions, design, implementation, quality checks, and change management done collaboratively at each stage.

Task 1: Design and Implement the Database and Data Access Layer using Class Library & EF Core

Learning outcomes assessed:

- Use object-oriented techniques to design and implement custom database entity classes
- Manage a multi-tier application which uses custom entity classes to access and manipulate a database
- Propose an appropriate technology for a particular problem

To build a comprehensive Equipment Rental Management System, you need to design your entities in a way that satisfies the functional and business requirements of the project.

Suggested Database entities *:

Entity	Entity Description
User	Stores information about system users, including their name, email, and user role. A user can be an administrator, rental manager, or customer.
Category	Stores rental equipment categories such as Power Tools, Cameras, Construction Equipment, Event Supplies, etc.
Equipment	Represents rental equipment, including name, description, category, rental price, availability status (available, unavailable, under maintenance, etc.), and condition status (new, good, damaged, etc.), etc..
Rental Request	A request raised by a customer to rent equipment. It includes details such as equipment ID, rental start date, return date, total cost, and status (Pending, etc.), etc..
Rental Transaction	Stores approved rental records, including assigned equipment, customer details, actual rental start (or pickup) date, return date, rental period, rental fee, deposit, and payment status (Paid, Pending, Overdue) , etc..
Return Record	Tracks returned equipment, including actual return date, return condition (e.g., good, damaged, lost), late return fees (if applicable), and any additional charges, etc.. *Note: this could be merged with the Rental Transaction or treated as a separate entity based on your own design assumptions*
Feedback	Containing notes (or ratings) from customers about their rented items, with date, time, user, and comment text, etc..
Document	Stores uploaded documents related to rentals, such as rental agreements, ID verifications, or damage reports, including file name, user, upload date, file type, and storage path (or BLOB data) , etc.. While some rentals do not need documents upload, some types of rentals do and thus the feature would be useful to track such documents.
Notification	Stores system-generated notifications for users, including message content, type (e.g., rental approval, return reminder), and status (read/unread), etc..
Log	Stores system logs, including user actions, exceptions, timestamps, affected data, and source (web or desktop app), etc.. *Note: This can be split into separate logs for errors and audit trails if needed.*
Other	You may add additional entities to ensure a fully functional system and a normalized database.

**This list is provided as a baseline example. As the lead designer, and you may need to evaluate what additional tables, entities, columns, etc. you need, and how to optimize your database and allow for a comprehensive, secure, and an easily maintainable system. (Modify as needed).*

You can follow either Database First or Code First approaches. In either case, make sure your database is well-designed, normalized, and includes auto-incremented primary keys, foreign keys, and correct relationships between tables.

For the user and role entities, you can design your own tables ensuring no passwords are stored in clear text. The recommended approach, however, is to use .NET Membership tables and use it to authenticate users on both your desktop and web applications.

Class Library project:

To develop the data access layer, create a Class Library project. The class library should model the database and allow access to it through Entity Framework Core Object-Relational Mapping Framework (OR/M).

Your Class Library should include:

1. Domain Classes with database tables mapping, annotated properties, and navigational properties.
2. Database Context with database connection that acts as a gateway to the Database.
3. DBSet collections which act as Database Tables

Task 2: Implement the ASP.NET Core MVC Web Application

Learning outcomes assessed:

- Design and implement web applications that access and manipulate a database
- Manage a multi-tier application which uses custom entity classes to access and manipulate a database
- Propose an appropriate technology for a particular problem

The ASP.NET Core Web Application should be the main interface for anyone who want to use this system. Each role will have their own permissions inside the Web Application, implemented through Authentication and Authorization (Preferably, using .NET Membership).

List of features & requirements *:

Feature	Requirements
Home	**Access Control:** • Available for guest access **Functionality:** 1. None. Includes general description of the system, its purpose, features, modules, etc. (informational only). The home page is the landing page for your system. Include images, tagline, promotional content, etc. to attract the visitor to explore your web application.
Authentication and Authorization	**Access Control:** • Anyone can access the registration and login pages. **Functionality:** 1. **Database Seeding:** An admin user should be seeded (inserted) in the database for elevated privileges

	2. **User Registration**: Users should be able to register themselves (as normal users/customers only) by providing their name, email, and password. 3. **User Login:** Users should also be able to login to the system using their email and password. 4. **User Profile:** Users should have a profile page to view and update their information or change their password.
Administration	**Access Control:** • Only Administrators should have access to the Administration pages **Functionality:** 1. **View Audit Trails / Logs:** The admin should be able to see audit trails and logs of the system 2. **(Optional feature):** The admin should be able to manage other users' roles, reset passwords, change information, etc.
Categories Management	**Access Control:** • Listing, searching, and reading Category information are allowed for all authenticated users. • Create, Update, and Delete categories are only allowed for the Administrator **Functionality:** 1. **List, Search, and Filter:** A user should be able to search and list all categories. Or, to view equipment filtered by category. 2. **Manage Categories:** The Administrator should be able to create, manage, and update categories.
Equipment Management	**Access Control:** 3. **Customers** have read access to all equipment. 4. **Managers and Administrators** can create and manage equipment details. **Functionality:** 5. **List, Search, and Filter:** Users should be able to search and list all equipment. 6. **Manage Equipment:** The administrator and managers should be able to create and manage all equipment in the system
Rental Requests	**Access Control:** • All customers should be able to create Rental Requests • Customers should have Read & Limited Update access to all requests they create (their own requests only). • Managers can update rental requests and change their status. **Functionality:**

	1. **List, Search, and Filter:** Customers should be able to search and view all rental requests they create. 2. **Create Rental Request:** Customers should be able to create rental requests and fill in its information. 3. **Update Rental Request:** Managers should be able to update all requests' information and status at all times. Customers should be able to update request description (only if the request has not been approved/confirmed/picked-up yet).
Rental Transaction	**Access Control:** • Customers should be able to view their rental transactions (only transactions involving them) • Administrator and Managers can update rental transactions **Functionality:** 1. **List, Search, and Filter:** Customers should be able to search and view all rental requests they create. 2. **Manage Rental Transaction:** Managers should be able to create and manage rental transactions. 3. **Manage Files:** The manager should be able to view, add, and delete files related to each transaction. 4. **View Files:** The customer should be able to view all files related to their own rental transactions.
Return Record	**Access Control:** • All customers should be able to view their return records (only records involving them) • Administrator and Managers can create and manage all return records **Functionality:** 1. **List, Search, and Filter:** Customers should be able to search and view all their return records. 2. **Manage Rental Transaction:** Managers should be able to create and manage return records.
Feedback	**Access Control:** • All authenticated users can view feedback comments or ratings • All customers should be able to write feedback or provide a rating for their rented items (only records involving them) • Administrator and Managers can hide any comment from public view **Functionality:** 1. **View Feedback:** Customers should be able to view feedback comments or ratings related to an equipment 2. **Create Feedback:** Customers should be able to write feedback about equipment they had rented before.

	3. **Manage Feedback:** Managers should be able to show/hide feedback text in case it wasn't appropriate.
Notifications	**Access Control:** • All authenticated users: All users in any roles will have access to the notifications sent specifically to them **Functionality:** **1. Receive Notifications:** o The customer should receive a notification when any information or status in his requests are updated 2. **View Notifications:** Notifications list ordered descending by date, showing status (read/unread) 3. **Notification Status:** When the user or manager opens the notification, its status should change from unread to read
Monitoring and Reporting	**Access control:** • Category dashboard should only be accessible to the Administrator and the Manager of each category **Functionality:** 1. **Dashboard:** Administrators and managers should have access to a dashboard, organized by equipment status, categories, etc., showing statistical summary and aggregated values about the equipment. For example: o Number of requests pending vs completed o Number of overdue requests o Number of requests per category o Damaged equipment report o Financial summary o Etc. *Note: The above is given as examples only. You need to decide what would be the best Dashboard information and KPIs for your system.*
Other Features	1. **Logging and Audit Trails:** The system should log any exception, or important user action in the database for audit purposes. This data will be important for the admin's Log page. *Note: Due to the complexity of the system, you can choose certain entities only to activate Change Tracking for, to include in your logging or audit trails but make sure they are properly highlighted in your documentation or your system. For example: Login attempts, exceptions, and a few entities change logs to show how you would achieve this.* 2. **Other:** You can add any other requirements you think is necessary to make the system more comprehensive in solving the business problem.

**This list is provided as a baseline example, without indicating how the workflow or process should be implemented. The list is incomplete but is only intended to provide a starting point. You are expected to improve upon it to make your system fully usable and user-friendly. (Modify as needed)*

Task 3: Implement the Windows Forms Project

Learning outcomes assessed:
• Design and implement desktop applications that access and manipulate a database • Manage a multi-tier application which uses custom entity classes to access and manipulate a database • Propose an appropriate technology for a particular problem

The Windows Forms Application should serve as a manager and staff tool to streamline day-to-day operations and provide real-time insights. It will complement the ASP.NET Core MVC Web Application by focusing on specific tasks that are better suited for a desktop environment.

List of features & requirements *:

Feature	Requirements
Authentication and Authorization	**Login:** Users should also be able to login to the system using the same email and password they use for the web application. Only **Administrators and Managers** are expected to use the Windows Desktop Form Application.
Rental Requests Quick View	**Access Control:** • Listing, searching, and reading Rental Requests related information is allowed for all authenticated users (Managers and Administrators). • Updating Rental Requests related information should be allowed for all authenticated users (Managers and Administrators). **Functionality:** 1. **List, Search, and Filter:** A user should be able to search and list all rental requests with their status 2. **Manage Rental Requests:** A user should be able to manage and update rental requests (i.e. change status, approve or reject, etc.) 3. **View Transaction and Return Records:** A user should be able to view the rental transaction and return record for each confirmed request.
Equipment Checkout (Rental Transaction)	**Access Control:** • Listing, searching, and reading Rental transaction information is allowed for all authenticated users (Managers and Administrators). • Creating and updating transactions, should be allowed for all authenticated users (Managers and Administrators). **Functionality:**

	1. **List, search, and filter** all transactions, or transactions related to an equipment or rental request 2. **Create and Manage Transactions:** A user should be able to create and manage a rental transaction 3. **(Optional):** Upload files related to the rental transaction *Note: while the file upload feature is mainly for the web system, it is optional if you wish to include it in the WinForm app as well*
Equipment Checkin (Return Record)	**Access Control:** • Listing, searching, and reading Return Records information is allowed for all authenticated users (Managers and Administrators). • Creating and updating Return Records, should be allowed for all authenticated users (Managers and Administrators). **Functionality:** 1. **List, search, and filter** all return records, or records related to an equipment or rental request 2. **Create and Manage Returns:** A user should be able to create and manage a return record related to rental transactions and equipment
Equipment Information	**Access Control:** • Listing, searching, and reading Rental related information is allowed for all authenticated users (Managers and Administrators). • Updating Equipment related information is allowed for all authenticated users (Managers and Administrators). **Functionality:** 1. **List, Search, and Filter:** A user should be able to filter, search and list all equipment with their information, status, and availability (rented, available, etc.). 2. **(Optional):** user can create or update the Equipment information directly from the Form Application.
Dashboard	**Access Control:** • Listing, searching, and reading Rental related information is allowed for all authenticated users (Managers and Administrators). • Updating Equipment related information is allowed for all authenticated users (Managers and Administrators). **Functionality:** 1. **View Equipment Dashboard:** The administrator as well as the managers can view a dashboard with relevant information about their equipment

	2. **View Rentals Dashboard:** The administrator as well as the managers can view a dashboard with relevant information about their rentals (highlighting aggregated values, as well as any overdue returns, etc.)
Other Features	1. **Logging and Audit Trails:** The system should log any exception, or important user action in the database for audit purposes. This information will help you create the Admin Logs page in the web application. *Note: Due to the complexity of the system, you can choose certain entities only to activate Change Tracking for, to include in your logging or audit trails but make sure they are properly highlighted in your documentation or your system. For example: Login attempts, exceptions, and a few entities change logs to show how you would achieve this.* 2. **Other:** You can add any other requirements you think is necessary to make the system more comprehensive in solving the business problem.

**This list is provided as a baseline example, without indicating how the workflow or process should be implemented. The list is incomplete but is only intended to provide a starting point. You are expected to improve upon it to make your system fully usable and user-friendly. (Modify as needed)*

Quality Standards:

Marks will be awarded for working code, but in addition to that marks will also be awarded for Code and GUI quality:

Code Quality

- Code uses the Class Library to maximise maintainability.
- The code of the applications is well-structured, organized, and easy to read and understand.
- Methods are used to eliminate repeated code.
- Class level variables are used to reduce repeated code.
- The web application views are strongly typed.
- Code is as efficient as possible to achieve the task.
- The number of database operation calls is minimized, and data is persisted where needed.
- Variables and controls have meaningful names.
- Code is commented meaningfully.
- Exception Handling/Validation is implemented.
- The system should be secure, and data should be protected from unauthorized access.

GUI Quality

- The system should be user-friendly, intuitive, and easy to use interface.
- The user interface provides all necessary functionalities.
- The web application should be responsive and work efficiently on various devices and browsers.
- Finding/Selecting data is enabled effectively.

- Filtered results are effectively displayed.
- Errors are minimised.
- Feedback is shown to the user upon successful or failed actions.
- Selected data populates related controls effectively.
- User choices are 'remembered' between form/page navigation events.
- Selected data populates related forms/pages effectively.

Note: 10% of the marks in each Application are awarded for Creativity/Innovation/Additional Functionality

For the 10%, think about real-world scenarios and edge cases:

1. **Handling Overlapping Rentals and Availability:** Consider how the system will manage rental requests for the same equipment with overlapping dates. Should the system automatically reject overlapping requests, or should it allow managers to manually approve one request over another? How will you ensure that equipment availability is accurately tracked and updated throughout the rental lifecycle?
2. **Rental Lifecycle and Workflow Design:** Think about when the rental officially starts. Should it be when the request is approved, or when the customer physically receives the item? How will the system handle cases where customer does not pick up the equipment (e.g., a no show)? Design a workflow that ensures smooth transitions between rental requests, transactions, and returns.

The 10% allocated to the marking criteria is for showing critical thinking about the process or workflow to achieve a creative solution, in addition to adding extra features to improve on the baseline examples given, and showing UI/UX skills and delivering a working and proof of concept that can be used in the real-world.

Deliverables:

You need to deliver the following items as part of your project:

1. **Data Access Layer:**
    - **Database:**

      You need to create a normalized database to store data related to the entities. The database should be created using Microsoft SQL Server (either from code first or database first using SQL Server Management Studio SSMS). You should export the full database including the schema and seeding data as a (.sql) script file which includes:
        - The database schema.
        - The SQL script to create the database and all tables.
        - The SQL script to populate the database with test data.

      Note: Your database must include tables used for authentication and authorization. If those are created in a separate database, then it must be submitted as well as a separate (.sql) script file.

- **Class Library:**

  You need to create a Class Library project that models your Database using Entity Framework Core. Your class library includes:
  - EFCore Packages.
  - Domain Classes with property annotations.
  - Database context with database connection string and DBSet collections.

2. **ASP.NET Core MVC Application:**

   You need to create an ASP.NET Core MVC application that provides a user interface to perform the tasks mentioned in the functional requirements above under ASP.NET Core MVC heading. The application should be developed using C# and .NET 5 or 6. Deliverables include:
   - The source code of the application.
   - The compiled executable of the application (which is already part of the Visual Studio solution folder).

3. **Windows Forms Application:**

   You need to create a Windows Forms application that provides a user interface to perform the tasks mentioned in the functional requirements above under Windows Form heading. The application should be developed using C# and .NET 5 or 6. Deliverables include:
   - The source code of the application.
   - The compiled executable of the application (which is already part of the Visual Studio solution folder).

4. **Project Document (with Screenshots)**

   You need to create a Project Document following the **template** provided, that includes:

   - The database design diagram, with short explanation or justification of the entities and relationships in case any assumptions were made.
   - A sample username and password for each user role in your application for testing and demonstration purposes.
   - Screenshots of the working ASP.NET Core Web Application showing different scenarios as described in the requirements and using different roles, and highlighting the approach, any additional components added, and any third-party libraries used.
   - Screenshots of the working Windows Forms Application showing different scenarios as described in the requirements and using different roles, and highlighting the approach, any additional components added, and any third-party libraries used.
   - Optional: You can submit a link to your group's Git-based repository (while providing necessary access), or submit a report or screenshot from the platform (i.e. Github) showing commits and contribution.

   The documentation is required to evaluate your database design, test your application, and clarify your approach. It highlights the different components of your project, your quality standard, and your creative approach to solve the given problem.

Marking Criteria:

The marking criteria for the project will be as follows:

Database:

- Does the database tables design satisfy the requirements?
- Is the database design normalized and efficient?
- Are all entities and relationships correctly modelled/implemented?
- Are naming conventions for tables and columns clear? Does the database utilize auto incrementing columns, NOT NULL for required columns, and efficient column types?
- Is the database script generated and submitted with enough data to support the application testing?

[20 marks]

Class Library:

- Are all required EF Core packages installed?
- Are all entities modelled correctly into domain classes with correct types and annotations?
- Are all entities relationships correctly modelled with Navigational Properties?
- Is the DBContext class present with DBSet collections and correct Connection String syntax and placement within the project?
- Are there references included from the other projects to the Class Library project or its DLL file?

[10 marks]

ASP.NET Core MVC Application:

- Functionality (70%):
  - Does the system meet all the functional requirements mentioned?
  - Does the system work correctly and efficiently?
- User Interface (10%):
  - Is the user interface intuitive and easy to use?
  - Does the user interface provide all necessary functionalities with proper feedback to users?
  - Is the user interface clean and responsive?
  - Is the look and feel consistent across the application (fonts, button sizes, etc)?
  - Does the interface follow the GUI quality standards?
- Code Quality (10%):
  - Is the code of the applications well-structured, organized, and easy to read and understand?
  - Are coding best practices followed, such as naming conventions, commenting, and error handling?

    - Does the code follow the code quality standards?
- Creativity, Innovation, and Additional Functionality (10%)
    - Are there creative and innovative enhancements within the application (for example: in terms of user interface, design, architecture, security, testing, or deployment) to make it more user friendly, robust, efficient, reliable, reusable, secure, scalable, maintainable, or useful in a bigger context?
    - Is there additional functionality added based on the analysis of the baseline features and requirements given that is useful, logical, and helps better solve the given business problem?

Note: In addition to evaluating your work by means of code review and live functionality testing, the system screenshots you will submit within your project document will be referred to when evaluating any of the above which you can use to clarify your approach.

[35 marks]

Windows Form Application:

- Functionality (70%):
    - Does the system meet all the functional requirements mentioned?
    - Does the system work correctly and efficiently?
- User Interface (10%):
    - Is the user interface intuitive and easy to use?
    - Does the user interface provide all necessary functionalities with proper feedback to users?
    - Is the look and feel consistent across the application (fonts, button sizes, etc)?
    - Does the interface follow the GUI quality standards?
- Code Quality (10%):
    - Is the code of the applications well-structured, organized, and easy to read and understand?
    - Are coding best practices followed, such as naming conventions, commenting, and error handling?
    - Does the code follow the code quality standards?
- Creativity, Innovation, and Additional Functionality (10%)
    - Are there creative and innovative enhancements within the application (for example: in terms of user interface, design, architecture, security, testing, or deployment) to make it more user friendly, robust, efficient, reliable, reusable, secure, scalable, maintainable, or useful in a bigger context?
    - Is there additional functionality added based on the analysis of the baseline features and requirements given that is useful, logical, and helps better solve the given business problem?

Note: In addition to evaluating your work by means of code review and live functionality testing, the system screenshots you will submit within your project document will be referred to when evaluating any of the above which you can use to clarify your approach.

[35 marks]

Task Allocation and Group Work Guidelines:

- Typically, a team leader must be identified who will manage the work, interactions, and direction.
- There are distinct tasks presented within the instructions document. A database designer and developer, Form developer, and Web Developer. Allocate the tasks based on your own team's strengths and the tasks' size.
- A typical allocation is: 1 team member for the database design and implementation as well as Class Library creation, 1 team member for the Form design and implementation, and 2-3 team members for the Web Application, divided by role as back-end/front-end, or by system feature.

Create a healthy group environment:

- Start on time.
- Agree on preferred communication platform.
- Meet at least once a week.
- Practice respect for yourself and others.
- Come prepared to do your part.
- Be a good listener.
- No put-downs.
- Make sure everyone gets a chance to contribute or speak.
- Accept constructive criticism gracefully.
- Critique ideas, not people.
- Stay on task.
- Discuss changes and their impact on each component.
- No interruptions; let people finish talking.
- Ask for help when you're confused about what to do.
- Help others when you can.
- Do your fair share of the work.
- Review others' work

Note: The standard procedure to plagiarism or dealing with group members not actively responding or participating in the group work will apply, which could include separation from the group, failing the group project component, or any other actions based on the policy. In case of group conflicts, discuss amongst your group first, and escalate to your tutor if required.

Conclusion:

The **Equipment Rental System** project is an opportunity for you to showcase your skills in database design and modelling, software development, and project management. It is an ambitious project that requires attention to detail, creativity, and teamwork.

The list of entities, features, and requirements given in the project instructions is provided as a baseline with the minimum requirements to start from. However, it is encouraged that you think outside the box, research more about this problem with its different approaches and theories, and create something that solves it in an innovative way with a production-ready quality. By the end of the project, you will have a fully functional system that enables companies to streamline their equipment rental operations. Good luck!