

CPT373 / COSC2363 Assignment 1 Specification

Deadline	Sunday, 03/07/2022 11:59pm AEST
Total marks	25 marks
% Allocated to this assignment	25%
Assignment mode	In a group of 2 OR individually
To be submitted via	Canvas

1.1 IMPORTANT

- a. You **MUST** submit your code via Canvas. See section 1.9 *Submission Procedure* for more information.
- b. You will be marked on the use of GitHub and development process. During Week 1 you will be emailed an invitation to join the **rmit-wdt-sp2-2022** GitHub Organisation, you must accept the invitation to join the organisation.
 - You must accept the invitation using a GitHub account that has been registered with your RMIT student email address.
 - Using a personal GitHub repository and not being a part of the **rmit-wdt-sp2-2022** GitHub Organisation **will lead to ZERO for the whole assignment.**
 - The GitHub repository for this assignment must be **private** and named as: **<student-id>-a1** for example **s3123456-a1**

If working in a group include **both** student IDs in the repository name as: **<student-id>-<student-id>-a1** for example **s3123456-s3654321-a1**
 - Include the URL of your GitHub repository in the readme file. The URLs for the repositories shown in the examples above would be:

<https://github.com/rmit-wdt-sp2-2022/s3123456-a1>
<https://github.com/rmit-wdt-sp2-2022/s3123456-s3654321-a1>
- c. You **MUST** make a commit at the beginning of the assignment period. You must also commit on a regular basis and make use of branches. Branches must represent the implementation of specific features.
- d. All data must be saved to an **Azure Microsoft SQL Server** database, you will be given your username and password during Week 2. Email Matthew Bolger at matthew.bolger2@rmit.edu.au if you have not received an invite to join the GitHub organisation or have not received your SQL Server credentials. **Using any other database or files to save data will lead to ZERO for the whole assignment.**
- e. In an advanced elective like this we place importance on code quality and the development process.
- f. Do **NOT MAKE ASSUMPTIONS** regarding the specifications. When in doubt post a question on Canvas or email us.
- g. The marks will be weighted according to your contribution within the group.
Please submit the contribution form.

1.2 PLAGIARISM

All assignments will be checked with plagiarism-detection software; any student found to have plagiarised would be subject to disciplinary action. Plagiarism includes:

- **CONTRACT CHEATING:** Paying someone to do your work.
- Posting assignment questions (in full or partial) on external forums, reddit, etc...
- Submitting work that is not your own or submitting text that is not your own.
- Copying work from / of previous / current semester students.
- Allowing others to copy your work via email, printouts, social media, etc...
- Sending or passing your work to your friends.
- Posting assignment questions on technical forums to get them solved.

A disciplinary action can lead to:

- A meeting with the disciplinary committee.
- A score of zero for the assignment.
- A permanent record of copying in your personal university records and / or
- Expulsion from the university, in some severe cases.

All plagiarism will be penalised. There are no exceptions and no excuses. You have been warned. For more details, please read RMIT's page on Academic Integrity at:

<https://www.rmit.edu.au/students/my-course/assessment-results/academic-integrity>

1.3 Scope

The aim of this assignment is to develop a menu-based console application for Internet Banking with **.NET 6.0** written in **C#** using **Visual Studio 2022** with an **Azure Microsoft SQL Server** database. Database access is to be implemented with **ADO.NET** and **SQL** using the **Microsoft.Data.SqlClient** package as the provider.

Use of an ORM is not allowed at this stage, you must only use SQL for database related operations.

You will extend the work done for this assignment into Assignment 2.

1.4 Overview

MCBA (Most Common Bank of Australia) has hired you to design their Internet Banking system. It is a simulated banking application. When complete, the system will be able to:

- Check balances & transaction history.
- Simulate transactions such as deposits and withdrawals.
- Transfer money between accounts.
- Schedule payments (**Assignment 2**).
- Modify a personal profile (**Assignment 2**).
- Perform administrative tasks (**Assignment 2**).

The system pre-loads customer, account, and login information from provided web-services. A **REST** call to these services is required to download **JSON** data which is then loaded into the backend database.

You will be **provided a SQL script** file called **CreateTables.sql**. Please make sure you execute this script on your database. This script is only a suggestion, you can modify the database structure, for example changing or adding constraints, default values, tables, fields, etc... however an unnormalized database with duplicated data / not well-defined entities / or too few tables will attract HEAVY penalty.

1.5 Business Rules

1. A **CustomerID** and **AccountNumber** are unique and will be exactly **4 digits** long.
2. A **LoginID** (used to hide customer information when logging in) are unique and are exactly **8 digits** long.
3. Only two types of accounts are allowed: **Savings** and **Checking**.
4. The minimum balance to open a savings account is **\$50** and **\$500** for a checking account.
5. The minimum balance allowed in a savings account is **\$0** and **\$300** for a checking account.
6. A user can have both a savings and checking account.
7. A user cannot have multiple saving accounts or multiple checking accounts.
8. The transaction types are:

Type	Description
D	Credit (Deposit money)
W	Debit (Withdrawal money)
T	Debit (Transfer money between accounts)
S	Debit (Service charge)

9. Service charges are applied on transactions as follows:

Fee Type	Fee (A\$)
ATM Withdraw	0.05
Account Transfer	0.10

10. **2 free transactions** are allowed **per account**.
11. **Negative** account balances are **not allowed**.
12. Transactions are **not allowed** an amount of **\$0**.
13. Transactions must record the time they occur in **UTC** into the database.
NOTE: When displaying the transaction time back to the user the **UTC** time must be converted into the **local time zone**.
14. All dates must be displayed in the **DD/MM/YYYY** format.
15. MCBA does not pay interest on any account.
16. No government taxes apply on any account.
17. Multiple fund transfers are not allowed.
18. Joint accounts are not allowed.

NOTE: Transfer transactions must have a different source account and target account, i.e., you cannot select the same account for both where money is transferred from and transferred to.

NOTE: When a transfer transaction occurs at least 2 transactions are added to the system not including the service charge. These transactions both have a transaction type of T, one for each account involved.

Consider the following transactions for transferring \$50 from account 4100 to account 4101:

	TransactionID	TransactionType	AccountNumber	DestinationAccountNumber	Amount	Comment	TransactionTimeUtc
1	10	T	4100	4101	50.00	Electricity Bill	2021-05-19 15:30:00.0000000
2	11	T	4101	NULL	50.00	Electricity Bill	2021-05-19 15:30:00.0000000

1.6 Tasks and Marks Allocation

Part 1: Loading Data and Login [5 marks]

a) [3 marks] Implement the pre-loading of data by making a **REST** call using **HTTP GET** to the provided customers web-service using the **JSON** returned to insert data into the database as appropriate. Generics should be used for the **JSON** deserialisation. The **REST** call should be implemented using the **System.Net.Http.HttpClient** class provided by **.NET**.

NOTE: Contacting the web-service should **only** occur if the system has no customer data, meaning if there are any customers present in the database then the web-service **should not** be contacted.

Customers web-service:

<https://coreteaching01.csit.rmit.edu.au/~e103884/wdt/services/customers/>

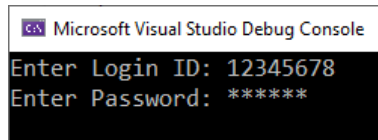
```
{
  "0": {
    "CustomerID": 2100,
    "Name": "Matthew Bolger",
    "Address": "123 Fake Street",
    "City": "Melbourne",
    "PostCode": "3000",
    "Accounts": [
      {
        "0": {
          "AccountNumber": 4100,
          "AccountType": "S",
          "CustomerID": 2100,
          "Transactions": [
            {
              "0": {
                "Amount": 100,
                "Comment": "Opening balance",
                "TransactionTimeUtc": "03/01/2022 08:00:00 PM"
              }
            ]
          }
        },
        "1": {
          "AccountNumber": 4101,
          "AccountType": "C",
          "CustomerID": 2100,
          "Transactions": [
            {
              "0": {
                "Amount": 600,
                "Comment": "First deposit",
                "TransactionTimeUtc": "03/01/2022 08:30:00 PM"
              },
              "1": {
                "Amount": 300,
                "Comment": "Second deposit",
                "TransactionTimeUtc": "03/01/2022 08:45:00 PM"
              }
            ]
          }
        }
      ]
    },
    "Login": {
      "LoginID": "12345678",
      "PasswordHash": "YBNbEL4Lk8yMEWxiKkGBeoILHTU7WZ9n8jJ5y8TNx0DAzNEFVsIVNRktiQV+I8d2"
    }
  }
}
```

When inserting customers and creating their accounts all the transactions from the web-service are to be transaction type D (Deposit). Additionally, the account's balance is to be set to the sum of all the account's associated transactions.

Refer to the **Dealing_with_Passwords.pdf** file for information on how to handle passwords and using the PasswordHash field.

b) [2 marks] Implement a login system. The system will first prompt the user to enter a **LoginID** and then a password. **The password should be masked while typing.**

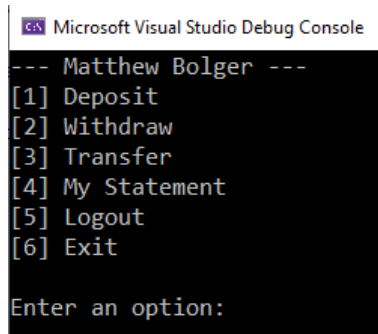
Consider the following example:



```
Microsoft Visual Studio Debug Console
Enter Login ID: 12345678
Enter Password: *****
```

There is **no** requirement to handle backspace whilst entering the password. It is recommended the masking of input be implemented using **Console.ReadKey(true);** to retrieve a character of input without displaying the input to the screen. Use additional logic to continue gathering input until enter is pressed.

If login fails an error message should be displayed. The system repeats the login process until successful. Once logged in a menu is displayed implementing the features of Part 2, for example:



```
Microsoft Visual Studio Debug Console
--- Matthew Bolger ---
[1] Deposit
[2] Withdraw
[3] Transfer
[4] My Statement
[5] Logout
[6] Exit

Enter an option:
```

Part 2: Essential Features [12 marks]

c) [3 marks] Implementation of ATM (Deposit & Withdraw) and Transfer features. The user should be able to transfer money between their own accounts and to accounts of other users.

d) [2 marks] Implementation of My Statements feature. My Statements allows the user to see the **current balance for a specified account** and list its transaction history. The transactions should be ordered by transaction time with the most recent transaction coming first and must be displayed in a paged manner, showing only 4 transactions at a time and allows the user to move to the next and previous pages as appropriate. The transaction time should display both the **date and time** information with the time portion including the hours and minutes.

e) [1 mark] Implementation of logout and exit features. Once logged out the user is returned to the initial login prompt. The console output should be cleared, and it should be possible to login as any user, i.e., logging in as a different user or even as the same user again.

Selecting the exit option ends the program. The program should display "Program ending." before terminating.

f) [3 marks] Implementation of two design patterns in the project and an analytical justification in the readme file. One of the design patterns must be the **Dependency Injection** design pattern. The remaining design pattern is up to you.

The analytical justification should cover the following points for both design patterns:

- Short summary of the design patterns.
- Briefly explain the purpose and advantage the design patterns offer to the project.
- Succinctly discuss your implementation of the design patterns.
- Identify where in the code, i.e., which file(s) and code within those file(s) are implementing the design patterns.
- Include any other important points.

IMPORTANT NOTE: Use of **Singleton** will **not** count as a design pattern.

NOTE: You must write the code for these design patterns. Simply using a design pattern that is already present in the framework is not sufficient.

g) [1 mark] Implement and use a class library in your project. You must justify its use and provide an explanation in the readme file.

IMPORTANT NOTE: Implementing a class library requires you to **create and use** your own class library project. Thus, the program will contain multiple projects and relative paths **must** be used. Simply using a NuGet package is not sufficient.

h) [2 marks] Use C#'s asynchronous keywords **async** and **await** in your implementation. You must justify the use of these keywords and provide an explanation of the advantages to using them in your readme file, e.g., how using these keywords change / benefit your design, method behaviour or program interaction / execution.

Part 3: Code Quality and Development Process [8 marks]

i) [4 marks] Code quality including comments, consistent indentation, code elegance, code repetition & object-orientation (encapsulation, coupling and cohesion).

Robust data validation* and exception handling**.

*Validation: Check user input including proper format and type. Ensure the business rules are not violated.

**Exception Handling: Your program should not crash under normal usage. For example, entering the input "abc" when a number is expected should not crash the program.

j) [4 marks] Effective use of version control and Trello.

Version control has the following guidelines:

- Effective use of a GitHub repository that demonstrates your development practises.
- Regular commits have been made throughout the development period.
- Commit messages should be meaningful, the messages can be short provided they are concise.
- Branches are used to represent each feature you are implementing; you may break a feature into additional sub-components.
- Branches should be given meaningful names.
- A minimum of 6 branches should be used in addition to the main branch.
- All branches should be merged into the main branch when completed.
- All branch commits must be checked and verified by your group partner prior to merging back into the main branch. If working in a group create and use GitHub Pull Requests to perform the merging of a branch into the main branch.

Trello has the following guidelines:

- Document your development and thought process over time as you build features of the software.
- The board should show the journey from the start of the project through to the end.
- Include milestones and features on the board. You may also include design, proof-of-concept, testing and bug-fixing tasks on the board if desired.
- You are allowed to use online templates.
- The boards should show sufficient details to the marker.
- Embed a minimum of 6 screenshots in your GitHub repository in a directory called Trello. The screenshots should be taken over time, for example adding a new screenshot every few days reflecting the project's progress. The file name should include the date in **year-month-day** format, for example **2022-05-30.png**. If multiple screenshots are taken on the same day, then include a timestamp using 24-hour format, for example **2022-05-30-11-30.png** and **2022-05-30-16-30.png**.

NOTE: Please read this online resource: <https://blog.trello.com/trello-board-best-practices>

NOTE: An example of a public Trello board used to track the Epic Games Store Roadmap can be found here: <https://trello.com/b/GXLC34hk/epic-games-store-roadmap>

1.7 Coding Standards

- Read the C# coding standard from the following website:
<https://docs.microsoft.com/en-au/dotnet/csharp/fundamentals/coding-style/coding-conventions>
- Remember that there are too many to be followed, implementing 6 to 10 of the standards will be a job well done.
- Do not force yourself to implement every object-oriented feature that you have learnt, use the features wisely and if needed.

1.8 Restrictions

No additional 3rd party plug-ins may be used in this assignment, without prior approval from the course instructor. Only use of the **Microsoft.Data.SqlClient**, **Newtonsoft.Json**, **Microsoft.Extensions.Configuration.Json** and **SimpleHashing** packages are permitted. You are required to write all your own classes following the principles of object-oriented design.

1.9 Submission Procedure

Each submission must include a readme file containing your full name, student ID, the URL to your GitHub repository, document your application, and any other relevant information. If you are working in a group, then please mention the details of your partner. The readme file should ideally be named **Readme.md** and be contained within the root of your repository.

Upload the solution / project, readme, and contribution form (if working in a group) as a single zip file to Canvas.

NOTE: You can include the contribution form within the repository if desired.

1.10 Late Submissions and Extension-Related Information

A penalty of 10% per day of the total marks for the assignment will apply for each day a submission is late, including both weekdays and the weekend. After 5 days, you will receive zero marks for the assignment. Email the course instructor matthew.bolger2@rmit.edu.au for extension related queries.