COSC2276/2277 Assignment part 2 specifications

Deadline	**Thursday 30.01.2020 (11:59 pm AEST)**
Face2FaceDemo	**Friday 31.01.2020** (schedule tba)
Test	**Friday 31.01.2020 (*during lab*)**
% allocated to this assignment	**20** **= 10% (assignment) + 10% (test)**
To be attempted (assignment)	**Individually or a group of 2**
To be attempted (test)	**Strictly individually**
To be submitted via:	**Canvas**

2.1 IMPORTANT:

a. This assignment has **two** separate components: face2face demo and a written test. Please make a note of the important dates mentioned above.

b. Demo will happen outside lab and the test will be scheduled in the lab. Without a face2face demo, no marks will be awarded. **You must submit your code via Canvas prior to the demo.**

c. You will be marked on the use of GITHUB | code development process | processes used and group dynamics | how you fare during demo. The final marks will be weighted according to your % contribution in group. If you cannot explain your own code during demo, you will receive a ZERO for the whole demo.

d. You will **not** be using your GITHUB for this assignment, you will be invited to be a part of GITHUB account created for you or your group.

e. The written test will be based upon the concepts used for assignment part 2. *This will include days 4- 9 lectures and tute/labs*. **This test will be held during your respective lab sessions**, a schedule (seating arrangement) for the test will be published, you must attend the lab where you are allocated for the test. Each of the group members will have to complete the test separately.

2.2 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
- CONTRACT CHEATING: getting someone else to write the test or attend demo
- 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.
- posting assignment questions (in full or partial) on external technical forums
- 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/student-essentials/assessment-and-exams/academic-integrity

2. 3 Scope:

The aim of this assignment is to develop an ASP.NET Core **Internet Banking website** with .NET Core 3.1 framework, using Microsoft Visual Studio 2019 with a Cloud SQL Server 2019 backend.

You will extend the work done for this assignment in part 3.

2.4 Overview:

NWBA (National Wealth Bank of Australasia) has hired you to design their Internet Banking website. It is a simulated banking application. When complete, one should be able to:

- Check balances
- Modify a personal profile
- Simulate transactions such as deposits and withdrawals
- Transfer money
- Schedule payments,
- Apply for loans (**needed for assignment 3 only**)
- Perform some administrative tasks (**needed for assignment 3 only**)

The following diagram depicts the logical architecture:

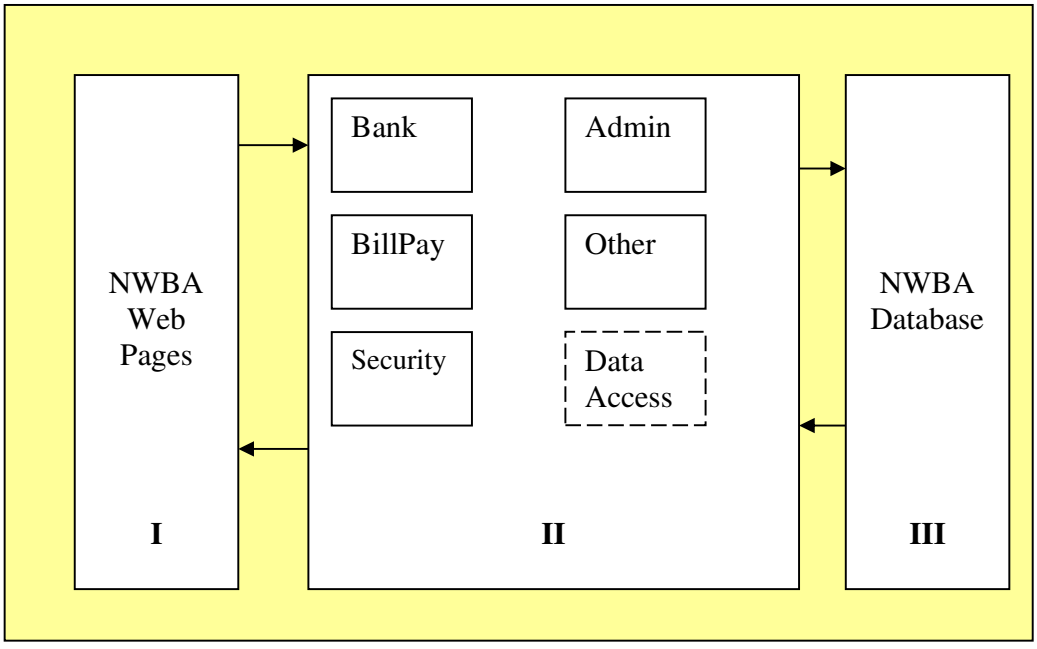

Figure 1

The architecture is a typical three-tiered, distributed application.

Layer I: User Interface (Presentation Layer) - will contain all the VIEW (and css, in case you decide to use style sheets) pages. *Lorem Ipsum is not allowed.* Please add meaningful content and images in the pages.

Layer II: Business logic/ middle tier/object layer: All the functionality must be present in this layer. You can use CONTROLLERS or business object * for this purpose. It is up-to-you how you want to write these class libraries. The following are needed:

`Bank`- this will handle all the customer and transaction information.

`BillPay`- this will handle the scheduled payment functionality.

`Security`- this will handle the user authentication and provide accounts with suitable capabilities (details later on)

`Other`- any extra functionality that you may want to add (Optional)

`Data Access`- This will provide all the functionality one needs to connect to database and retrieve information.

Layer III: Database/Data tier- this comprises of your Cloud SQL Server 2019 database (tables, etc.)

For database operations you are ONLY allowed to use Entity Framework (EF) Core framework. If your query is too complex and EF Core syntax does not support it, you may use an Embedded SQL, but you must justify use of any SQL during face2face demo. In a nutshell, do not resort to using SQL unless necessary.

**: Business object is a separate file where some piece of usable functionality is written. Instead of writing all application logic in controllers, you can call methods defined in business objects.*

Modified database

Here is a more detailed version of the database provided to you for assignment 1, feel free to modify this structure, however an unnormalised database with too few tables will attract HEAVY penalty.

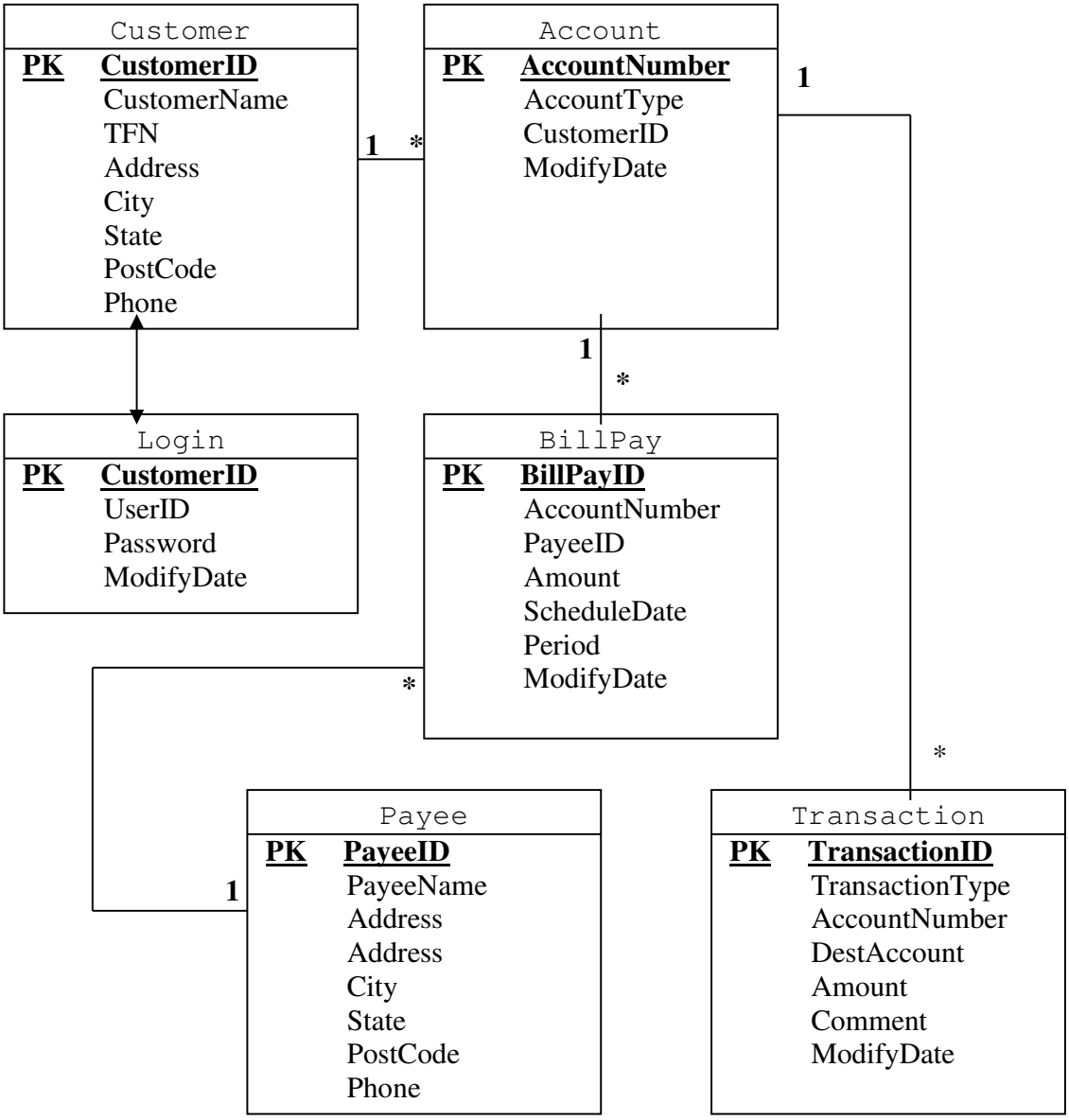

Figure 2

A brief description of these tables follows:

NOTE: All the format constraints must be enforced on the ASP.NET-side.

Customer Table

Field name	Data Type	Length	Description	Format	Allow Null
CustomerID	int	4	A unique ID for each customer		NOT NULL
CustomerName	nvarchar	50			NOT NULL
TFN	nvarchar	11	Tax File Number– this is just for identification purposes– *no tax implications*		
Address	nvarchar	50			
City	nvarchar	40			
State	nvarchar	20		Must be 3 lettered Australian state	
PostCode	nvarchar	10		Must be a 4 digit number	
Phone	nvarchar	15		Must be of the format: (61)– XXXX XXXX	NOT NULL

Login Table

Field name	Data Type	Length	Description	Format	Allow Null
CustomerID	int	4	Refers back to customer		NOT NULL
UserID	nvarchar	50			NOT NULL
Password	nvarchar	20		Must be stored in an encoded format	NOT NULL
ModifyDate	datetime	8	Last date/time		NOT

			the UserID and Password were modified		NULL

Account Table

Field name	Data Type	Length	Description	Format	Allow Null
AccountNumber	int	4	Auto-generated unique ID for the account; also the customer's actual account number		NOT NULL
AccountType	nvarchar	1	The type of account such as checking or savings account	"C" or "S"	NOT NULL
CustomerID	int	4	Customer's unique ID		NOT NULL
ModifyDate	datetime	8	Last date/time record was modified		NOT NULL

BillPay Table

Field name	Data Type	Length	Description	Format	Allow Null
BillPayID	int	4	Auto-generated unique ID for each billpay created		NOT NULL
AccountNumber	int	4	Source account number to withdraw funds from		NOT NULL
PayeeID	int	4	Payee to send payment to		NOT NULL
Amount	money	8	Amount of funds to be taken from the account		NOT NULL
ScheduleDate	datetime	8	Next schedule date for transaction to occur		NOT NULL
Period	nvarchar	1	How often will this occur?	Monthly (M), Quarterly (Q), Annually (Y) or	NOT NULL

				Once off (S)	
ModifyDate	datetime	8	Last date/time record was modf.		NOT NULL

Payee Table

Field name	Data Type	Length	Description	Format	Allow Null
PayeeID	int	4	Auto-generated unique ID		NOT NULL
PayeeName	nvarchar	50			NOT NULL
Address	nvarchar	50	Payee's address		
City	nvarchar	40			
State	nvarchar	20		Must be 3 lettered Australian state	
PostCode	nvarchar	10		Must be a 4 digit number	
Phone	nvarchar	15		Must be of the format: (61)- XXXX XXXX	NOT NULL

Transaction Table

Field name	Data Type	Length	Description	Format	Allow Null
TransactionID	int	4	Auto-generated unique ID for each transaction type		NOT NULL
TransactionType	nvarchar	1	Type of transaction taking place	Refer to code table on next page	NOT NULL
AccountNumber	int	4	Source account number		NOT NULL
DestAccount	int	4	Destination account- used for transfers		
Amount	money	8	Amount of credit or debit		
Comment	nvarchar	255	Any comments the banker added to the transaction		

ModifyDate	datetime	8	Last date/time record was modified		

Transaction types:

D = Credit (Deposit money)
W = Debit (Withdrawal)
T = Debit (Transfer)
S = Debit (Service Charge)* - see business rules for more details on this
B = Debit (BillPay)

Thus in a nutshell,

Customer table is used to store all the customer information. We will distinguish between bank customers and payees by creating a separate Payee table. Only the bank's own customers will be held in the Customers table.

Account table is used to store all the information about the different accounts that any customer may have.

Transactions table us used to track all credit and debit items. Credit items will include transactions such as a customer's deposit of funds, a bank-applied credit (i.e. if any). Debit items will include items such as paying out on a check written by a customer, a monthly service charge, transfer of funds, ATM withdrawals, and so on. Thus, for each credit add the amount to the balance and for each debit, subtract the funds from the account.

BillPay table is used to schedule automatic payments to a third party such as a telephone bill, electricity bill, or a home mortgage company. You can schedule these payments to be made monthly, quarterly, annually, or on a specific date, as a one-off payment.

Login table is used to store customer's username and encoded password so they can log in to the web site.

2.5 Business Rules:

Refer to assignment 1 specifications for these.

2.6 Tasks and Marks allocation:

Part A

a (5 marks) Every user must be presented with a login page (start page), which should have some introductory information about the bank and a form to log in. Also provide a sign off link. **If you wish to use Identity API, kindly delete the Login table from the database.**

Every logged in user to should first see the ATM page. This page mimics the ATM functionality. Each page must have a navigation bar, which should have the following links-

Deposit/Withdraw (ATM page) || My Statement (to display transaction history) || My Profile (Account management)

A rough (suggested) layout of the ATM page is as follows:

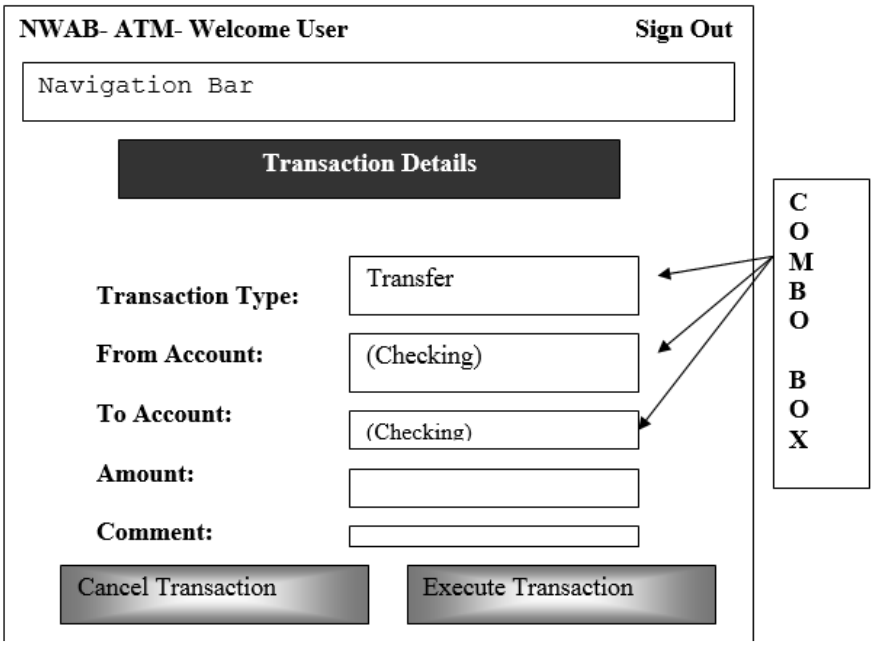

Figure 3

Make sure that **login** and **logout** are fully functional.

b (3 marks) My Statement page allows the user to see the current balance of a specific account and get a listing of all the transactions. User should first be presented with an option to choose the account type- checking or savings (drop-down box). Upon the selection of the type of account, the transactions must be displayed in a paged control (only 4 transactions per page).

c (2 marks) My Profile page should be used to modify user's information. It should have the form controls to edit information as- Name, Address, City, State, Post Code, Phone and TFN.

User should also be to change the password. Any changes made must be written back to the database.

d (5 marks) All the format constraints (mentioned in the table descriptions) must be enforced using ASP.NET Core.

Part B

e (10 marks) Add another link to the navigation bar- Pay Bills. Thus implement the bill pay option. When use clicks at this link, he/she is taken to a page where bill pays can be scheduled. A rough (suggested) layout of this page is as follows:

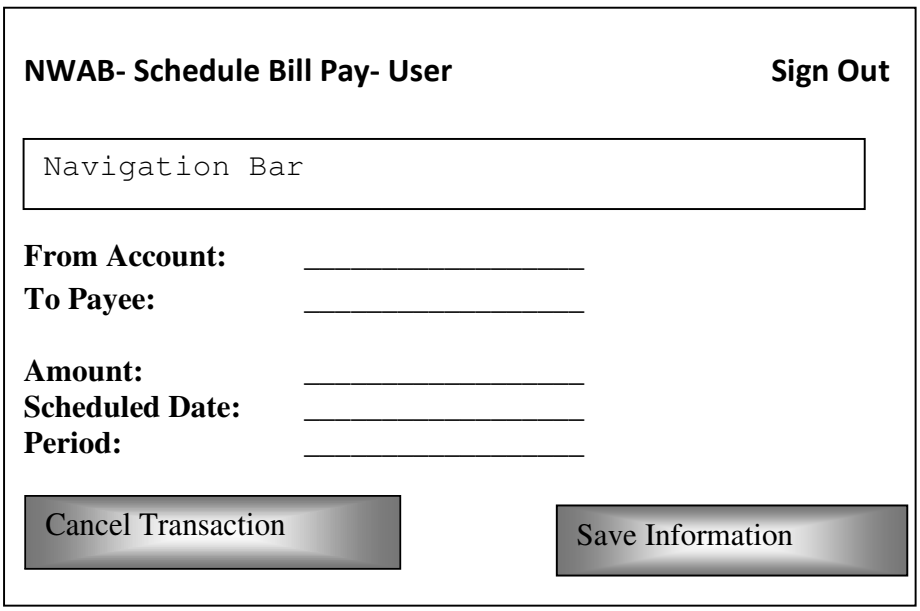

Figure 4

Implement another page, which displays all the scheduled bill pay information. This page should display the information in a grid-like manner, in which one the columns must have a modify button. A rough layout is shown as below:

	Payee	**Amount**	**Schedule Date**	**Period**
Modify	Telstra	$24.99	3/9/2020	One Time
Modify	RMIT	$1000	10/12/2020	Annual

Figure 5

Once the user clicks at the Modify button, he /she should be taken to form shown in figure 4.

f (10 marks) All of the above features must use a business objects. This essentially means that Controllers should not be bloated with logic behind the implementation. In order to get these marks, you must write a separate files for each of the features implemented above.

You must provide an analytical justification of the business objects used in README.txt file.

j (5 marks) Effective use of GitHub repository.

2.7 Coding Standards:

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

2.8 Restrictions:

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

2.9 Submission Procedure:

Each submission must include a README.txt file containing your full name, Student ID and any other relevant information (if you are working in a group, then please mention the details of your partner).

Zip the entire solution project folder (with README.txt file included) and submit it via Canvas as a single zipped archive.

2.10 Late submissions and Extension-related information:

A penalty of 10% per day of the total marks for each 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 that assignment. Email your lecturer (shekhar.kalra@rmit.edu.au) for extension related queries.