Development environment preparation

CRITERIA	MEETS SPECIFICATIONS
Create modules folders	1. Create a new project 2. Create "Application" folder 3. Create "Card" folder 4. Create "Terminal" folder 5. Create "Server" folder Note: To create a folder in Microsoft Visual Studio 1. In the solution explorer, right-click on the project name 2. Select Add 3. Select Folder 4. Give a name to that folder You should deliver a screenshot of the solution explorer that clarifies your folder structure.
Create .c and .h file for each module	1. In the "Application" folder create **app.c** and **app.h** files 2. In the "Card" folder create **card.c** and **card.h** files 3. In the "Terminal" folder create **terminal.c** and **terminal.h** files 4. In the "Server" folder create **server.c** and **server.h** files Note: To create a file into a folder in Microsoft Visual Studio: 1. In the solution explorer, right-click on the folder you want 2. Select Add 3. Select New Item 4. Select file type, .cpp or .h 5. If a .cpp extension, change the extension to .c 6. Give a name to that file You should deliver a screenshot of the solution explorer that clarifies files in each folder.
Add header file guard	1. In the **app.h** file add the header file guard 2. In the **card.h** file add the header file guard 3. In the **terminal.h** file add the header file guard 4. In **server.h** file add the header file guard You should deliver a screenshot for each .h file, the file name must appear in the screenshot, and the header file guard

Implement the card module

CRITERIA	MEETS SPECIFICATIONS

Fill in card.h file with functions' prototypes and typedefs	1. Use the following typedef as is:
	`typedef uint8_t* uint8ptr_t;`
	`typedef struct ST_cardData_t`
	`{`
	`uint8_t cardHolderName[25];`
	`uint8_t primaryAccountNumber[20];`
	`uint8_t cardExpiryDate[5];`
	`};`
	`ST_cardData_t;`
	
	`typedef enum EN_cardError_t`
	`{`
	`CARD_OK, WRONG_NAME, WRONG_EXP_DATE, WRONG_PAN`
	`EN_cardError_t;`
	
	2. Use the following prototypes as is:
	`EN_cardError_t getCardHolderName(ST_cardData_t *cardData);`
	`EN_cardError_t getCardExpiryDate(ST_cardData_t *cardData);`
	`EN_cardError_t getCardPAN(ST_cardData_t *cardData);`
	
	You should deliver a screenshot of your card.h file

Implement getCardHolderName function	1. This function will **ask** for the **cardHolder's name and store it into card data**.
	2. Cardholder name is **24 alphabetic characters string max and 20 min**.
	3. If the cardholder name is **less** or **more than 20 characters** or **more than 24** will return a `WRONG_NAME` error, else return `CARD_OK`.
	4. Test your function:
	• Create a new function `void getCardHolderNameTest(void);` to test all possible scenarios, happy case, and worst case scenarios.
	• Print all results of your test cases on the console window, use the following as a guide:
	`Tester Name: your name`
	`Function Name: getCardHolderName`
	`Test Case 1:`
	`Input Data:`
	`Expected Result:`
	`Actual Result:`
	`Test Case 2:`
	`Input Data:`
	`Expected Result:`
	`Actual Result:`
	`.`
	`.`
	`.`
	`Test Case n:`
	`Input Data:`
	`Expected Result:`
	`Actual Result:`
	
	5. You should deliver the test function as well as a screenshot of the results on the console.

Implement getCardExpiryDate function	1. This function will **ask** for the **card expiry date and store it in card data**.
	2. Card expiry date is **5 characters string** in the format "MM/YY" e.g "08/20".
	3. If the card expiry date is `NULL`, **less** or **more than 5 characters**, or has the **wrong format** will return the `WRONG_EXP_DATE` error, else return `CARD_OK`.
	4. Test your function:
	• Create a new function `void getCardExpiryDateTest(void);` to test all possible scenarios, happy case, and worst case scenarios.
	• Print all results of your test cases on the console window, use the following as a guide:
	`Tester Name: your name`
	`Function Name: getCardExpiryDate`
	`Test Case 1:`
	`Input Data:`
	`Expected Result:`
	`Actual Result:`
	`Test Case 2:`
	`Input Data:`
	`Expected Result:`
	`Actual Result:`
	`.`
	`.`
	`.`
	`Test Case n:`
	`Input Data:`
	`Expected Result:`
	`Actual Result:`
	
	5. You should deliver the test function as well as a screenshot of the results on the console.

Implement getCardPAN function	1. This function will **ask** for the card's **Primary Account Number and store it in card data**.
	2. PAN is **20 numeric characters string**, **19 character max**, and **16 character min**.
	3. If the PAN is `NULL`, **less** than **16** or **more** than **19 characters** will return the `WRONG_PAN` error, else return `CARD_OK`.
	4. Test your function:
	• Create a new function `void getCardPANTest(void);` to test all possible scenarios, happy case, and worst case scenarios.
	• Print all results of your test cases on the console window, use the following as a guide:
	`Tester Name: your name`
	`Function Name: getCardPAN`
	`Test Case 1:`
	`Input Data:`
	`Expected Result:`
	`Actual Result:`
	`Test Case 2:`
	`Input Data:`
	`Expected Result:`
	`Actual Result:`
	`.`
	`.`
	`.`
	`Test Case n:`
	`Input Data:`
	`Expected Result:`
	`Actual Result:`
	
	5. You should deliver the test function as well as a screenshot of the results on the console.

Explain your work	1. Record a video where you discuss each function you implemented in this module. 2. Explain and execute all test functions you made. 3. The video is **4 minutes maximum**. 4. You may record it in Arabic or English. 5. Recorded videos will not be accepted. 6. **All of the above are mandatory to pass this criterion.**

Implement the terminal module

CRITERIA	MEETS SPECIFICATIONS

Fill in terminal.h file with functions' prototypes and typedefs	1. Use the following typedef as is:
	`typedef struct ST_terminalData_t`
	`{`
	`float transAmount;`
	`float maxTransAmount;`
	`uint8_t transactionDate[11];`
	`};`
	`ST_terminalData_t ;`
	
	`typedef enum EN_terminalError_t`
	`{`
	`TERMINAL_OK, WRONG_DATE, EXPIRED_CARD, INVALID_CARD, INVALID_AMOUNT, EXCEED_MAX_AMOUNT, INVALID_MAX_AMOUNT`
	`EN_terminalError_t ;`
	
	2. Use the following prototypes as is:
	`EN_terminalError_t getTransactionDate(ST_terminalData_t *terminalData);`
	`EN_terminalError_t isCardExpired(ST_cardData_t *cardData, ST_terminalData_t *terminalData);`
	`EN_terminalError_t getTransactionAmount(ST_terminalData_t *terminalData);`
	`EN_terminalError_t isValidAmount(ST_terminalData_t *terminalData);`
	`EN_terminalError_t setMaxAmount(ST_terminalData_t *terminalData, float maxAmount);`
	`EN_terminalError_t isBelowMaxAmount(ST_terminalData_t *terminalData);`
	
	3. You should deliver a screenshot for your terminal.h file

Implement getTransactionDate function	1. This function will **ask** for the **transaction date and store it in terminal data**.
	2. Transaction date is **10 characters string** in the format **DD/MM/YYYY**. e.g 25/06/2022.
	3. If the transaction date is `NULL`, or **less** than **10 characters** or **wrong format** will return the `WRONG_DATE` error, else return `TERMINAL_OK`.
	4. Test your function:
	• Create a new function `void getTransactionDateTest(void);` to test all possible scenarios, happy case, and worst case scenarios.
	• Print all results of your test cases on the console window, and use the following as a guide:
	`Tester Name: your name`
	`Function Name: getTransactionDate`
	`Test Case 1:`
	`Input Data:`
	`Expected Result:`
	`Actual Result:`
	`Test Case 2:`
	`Input Data:`
	`Expected Result:`
	`Actual Result:`
	`.`
	`.`
	`.`
	`Test Case n:`
	`Input Data:`
	`Expected Result:`
	`Actual Result:`
	
	5. **Optional**
	The function will read the current date from your computer and store it into terminal data with the mentioned size and format.
	6. You should deliver the test function as well as a screenshot of the results on the console.

Implement isCardExpired function	1. This function **compares the card expiry date with the transaction date**.
	2. If the card **expiration date is before** the transaction date will return `EXPIRED_CARD`, else return `TERMINAL_OK`.
	3. Test your function:
	• Create a new function `void isCardExpiredTest(void);` to test all possible scenarios, happy case, and worst case scenarios.
	• Print all results of your test cases on the console window, and use the following as a guide:
	`Tester Name: your name`
	`Function Name: isCardExpired`
	`Test Case 1:`
	`Input Data:`
	`Expected Result:`
	`Actual Result:`
	`Test Case 2:`
	`Input Data:`
	`Expected Result:`
	`Actual Result:`
	`.`
	`.`
	`.`
	`Test Case n:`
	`Input Data:`
	`Expected Result:`
	`Actual Result:`
	
	4. You should deliver the test function as well as a screenshot of the results on the console.

Implement getTransactionAmount function	1. This function **asks** for the **transaction amount and saves it into terminal data**.
	2. If the transaction amount is **less than or equal to 0** will return `INVALID_AMOUNT`, else return `TERMINAL_OK`.
	3. Test your function:
	• Create a new function `void getTransactionAmountTest(void);` to test all possible scenarios, happy case, and worst case scenarios.
	• Print all results of your test cases on the console window, and use the following as a guide:
	`Tester Name: your name`
	`Function Name: getTransactionAmount`
	`Test Case 1:`
	`Input Data:`
	`Expected Result:`
	`Actual Result:`
	`Test Case 2:`
	`Input Data:`
	`Expected Result:`
	`Actual Result:`
	`.`
	`.`
	`.`
	`Test Case n:`
	`Input Data:`
	`Expected Result:`
	`Actual Result:`
	
	4. You should deliver the test function as well as a screenshot of the results on the console.

Implement isValidAmount function	1. This function **compares the transaction amount with the terminal max allowed amount**.
	2. If the transaction amount is **larger than the terminal max allowed amount** will return `EXCEED_MAX_AMOUNT`, else return `TERMINAL_OK`.
	3. Test your function:
	• Create a new function `void isValidAmountTest(void);` to test all possible scenarios, happy case, and worst case scenarios.
	• Print all results of your test cases on the console window, and use the following as a guide:
	`Tester Name: your name`
	`Function Name: isValidAmount`
	`Test Case 1:`
	`Input Data:`
	`Expected Result:`
	`Actual Result:`
	`Test Case 2:`
	`Input Data:`
	`Expected Result:`
	`Actual Result:`
	`.`
	`.`
	`.`
	`Test Case n:`
	`Input Data:`
	`Expected Result:`
	`Actual Result:`

Implement setMaxAmount function	1. This function **takes the maximum allowed amount and stores it into terminal data**.
	2. Transaction max amount is a **float number**.
	3. If transaction max amount **less than or equal to 0** will return the `INVALID_MAX_AMOUNT` error, else return `TERMINAL_OK`.
	4. Test your function:
	• Create a new function `void setMaxAmountTest(void);` to test all possible scenarios, happy case, and worst case scenarios.
	• Print all results of your test cases on the console window, and use the following as a guide:
	`Tester Name: your name`
	`Function Name: setMaxAmount`
	`Test Case 1:`
	`Input Data:`
	`Expected Result:`
	`Actual Result:`
	`Test Case 2:`
	`Input Data:`
	`Expected Result:`
	`Actual Result:`
	`.`
	`.`
	`.`
	`Test Case n:`
	`Input Data:`
	`Expected Result:`
	`Actual Result:`
	
	5. You should deliver the test function as well as a screenshot of the results on the console.

Implement isBelowMaxAmount function **(Optional)**	1. This function will **check if the PAN is a Luhn number or not**.
	2. For more about Luhn number, visit here to generate and check please refer to this Site.
	3. If the number is not Luhn number, will return `INVALID_CARD`, else will return `TERMINAL_OK`.
	4. Test your function:
	• Create a new function `void isValidLuhnNumberTest(void);` to test all possible scenarios, happy case, and worst case scenarios.
	• Print all results of your test cases on the console window, and use the following as a guide:
	`Tester Name: your name`
	`Function Name: isValidLuhnNum`
	`Test Case 1:`
	`Input Data:`
	`Expected Result:`
	`Actual Result:`
	`Test Case 2:`
	`Input Data:`
	`Expected Result:`
	`Actual Result:`
	`.`
	`.`
	`.`
	`Test Case n:`
	`Input Data:`
	`Expected Result:`
	`Actual Result:`
	
	5. If you are going to implement this function, please deliver the test function as well as a screenshot of the results on the console.

Explain your work	1. Record a video where you discuss each function you implemented in this module. 2. Explain and execute all test functions you made. 3. The video is **4 minutes maximum**. 4. You may record it in Arabic or English. 5. Recorded videos will not be accepted. 6. **All of the above are mandatory to pass this criterion.**

Implement the server module

CRITERIA	MEETS SPECIFICATIONS

Fill in server.h file with functions' prototypes and typedefs	1. Use the following typedef as is:
	`typedef enum EN_transState_t`
	`{`
	`APPROVED, INSUFFICIENT_FUND, INTERNAL_SERVER_ERROR`
	`EN_transState_t;`
	
	`typedef struct ST_transaction_t`
	`{`
	`ST_cardData_t cardHolderData;`
	`ST_terminalData_t terminalData;`
	`EN_transState_t transState;`
	`uint32_t transactionSequenceNumber;`
	`ST_transaction;`
	
	`typedef enum EN_accountState_t`
	`{`
	`RUNNING,`
	`BLOCKED`
	`EN_accountState_t;`
	
	`typedef struct ST_accountDB_t`
	`{`
	`float balance;`
	`EN_accountState_t state;`
	`uint8_t primaryAccountNumber[20];`
	`ST_accountDB_t ;`
	
	2. Use the following prototypes as is:
	`EN_transState_t recieveTransactionData(ST_transaction_t *transaction);`
	`EN_serverError_t isValidAccount(ST_cardData_t *cardData, ST_accountDB_t *accountReference);`
	`EN_serverError_t isBlockedAccount(ST_accountDB_t *accountReference);`
	`EN_serverError_t isAmountAvailable(ST_terminalData_t *terminalData, ST_accountDB_t *accountReference);`
	`EN_serverError_t saveTransaction(ST_transaction_t *transaction);`
	`void listSavedTransactions(void);`
	
	3. You should deliver a screenshot for your server.h file.

Implement server side account database	1. Create a global array `ST_accountDB_t` for the valid accounts database. `ST_accountDB_t accounts[255];` 2. Fill the array with the following valid data. 3. This array has **a maximum of 255 elements/transaction data**. 4. You can fill up to **10 different accounts for the sake of testing**. 5. Hard code the **primary account number** in the "**Testing the application**". 6. Example of a **blocked** account, its **state** is written `BLOCKED`, **balance**, "**PAN**"**NOTHINGINTERESTS**". 7. You should deliver a screenshot of your accounts database array with a minimum of at least 5 different accounts for the different tests cases. check all needed tests cases in the "**Testing the application**" section.
Implement server side transactions' database	1. Create a global array `ST_transaction_t`. 2. Fill in the array **initially with Zeros**. 3. This array has **a maximum of 255 elements/transaction data**. 4. You should deliver a screenshot of your transaction database array.

Implement recieveTransactionData function	1. This function will **take all transaction data and validate its data**. It contains all server logic.
	2. It checks the account details and amount available.
	3. If the account **does not exist** return `FRAUD_CARD`, if the **amount is not available** will return `DECLINED_INSUFFICIENT_FUND`, if the account is **blocked** return `DECLINED_STOLEN_CARD`, if **a transaction can't be saved** will return `INTERNAL_SERVER_ERROR`, else return `APPROVED`.
	4. It **updates the database with the new balance**.
	5. Test your function:
	• Create a new function `void recieveTransactionDataTest(void);` to test all possible scenarios, happy case, and worst case scenarios.
	• Print all results of your test cases on the console window, and use the following as a guide:
	`Tester Name: your name`
	`Function Name: recieveTransactionData`
	`Test Case 1:`
	`Input Data:`
	`Expected Result:`
	`Actual Result:`

```
Actual Result:
Test Case 2:
Input Data:
Expected Result:
Actual Result:
.
.
.
Test Case n:
Input Data:
Expected Result:
Actual Result:
```

6. You should deliver the test function as well as a screenshot of the results on the console.

Implement isValidAccount function

1. This function will **take card data** and validate **if the account related to this card exists or not**.
2. It checks if the **PAN exists or not** in the server's database (**searches for the card PAN in the DB**).
3. If the PAN doesn't exist will return `ACCOUNT_NOT_FOUND` and the account reference will be `NULL`, else will return `SERVER_OK` and return **a reference to this account in the DB**.
4. Test your function:
   - Create a test function `void isValidAccountTest(void);` to test all possible scenarios, happy-case, and worst-case scenarios.
   - Print all results of your test cases on the console window, and use the following as a guide:

```
Tester Name: your name
Function Name: isValidAccount
Test Case 1:
Input Data:
Expected Result:
Actual Result:
Test Case 2:
Input Data:
Expected Result:
Actual Result:
.
.
.
Test Case n:
Input Data:
Expected Result:
Actual Result:
```

5. You should deliver the test function as well as a screenshot of the results on the console.

Implement isBlockedAccount function

1. This function **takes a reference to the account into the database and verifies if it is blocked or not**.
2. If the account is **running** it will return `SERVER_OK`, else if the account is **blocked** it will return `BLOCKED_ACCOUNT`.
3. Test your function:
   - Create a test function `void isBlockedAccountTest(void);` to test all possible scenarios, happy-case, and worst-case scenarios.
   - Print all results of your test cases on the console window, and use the following as a guide:

```
Tester Name: your name
Function Name: isBlockedAccount
Test Case 1:
Input Data:
Expected Result:
Actual Result:
Test Case 2:
Input Data:
Expected Result:
Actual Result:
.
.
.
Test Case n:
Input Data:
Expected Result:
Actual Result:
```

4. You should deliver the test function as well as a screenshot of the results on the console.

Implement isAmountAvailable function

1. This function will **take terminal data and a reference to the account** in the database and **check if the account has a sufficient amount to withdraw or not**.
2. It checks if the transaction's amount is available or not.
3. If the transaction amount **is greater than the balance** in the database will return `LOW_BALANCE`, else will return `SERVER_OK`.
4. Test your function:
   - Create a test function `void isAmountAvailableTest(void);` to test all possible scenarios, happy-case, and worst-case scenarios.
   - Print all results of your test cases on the console window, and use the following as a guide:

```
Tester Name: your name
Function Name: isAmountAvailable
Test Case 1:
Input Data:
Expected Result:
Actual Result:
Test Case 2:
Input Data:
Expected Result:
Actual Result:
.
.
.
Test Case n:
Input Data:
Expected Result:
Actual Result:
```

5. You should deliver the test function as well as a screenshot of the results on the console.

Implement saveTransaction function

1. This function will **store all transaction data** in the transactions database.
2. It gives **the sequence number to a transaction**, **this number is incremented once a transaction is processed** into the server, you must check the last sequence number in the server to give the new transaction a **new sequence number**.
3. It saves any type of transaction, `APPROVED`, `DECLINED_INSUFFECIENT_FUND`, `DECLINED_STOLEN_CARD`, `FRAUD_CARD`, `INTERNAL_SERVER_ERROR`.
4. It will list all saved transactions using the `listSavedTransactions` function.
5. Assuming that the connection between the terminal and server is always connected, then it will return `SERVER_OK`.
6. Test your function:
   - Create a test function `void saveTransactionTest(void);` to test all possible scenarios, happy-case, and worst-case scenarios.
   - Print all results of your test cases on the console window, and use the following as a guide:

```
Tester Name: your name
Function Name: saveTransaction
Test Case 1:
Input Data:
Expected Result:
Actual Result:
Test Case 2:
Input Data:
Expected Result:
Actual Result:
.
.
.
Test Case n:
Input Data:
Expected Result:
Actual Result:
```

7. You should deliver the test function as well as a screenshot of the results on the console.

Implement listSavedTransactions function

1. This function **prints** all transactions found in the transactions DB.
2. Please follow the following format for only one transaction data:

```
##########################
Transaction Sequence Number:
transaction Date:
Transaction Amount:
Transaction State:
Terminal Max Amount:
Cardholder Name:
PAN:
Card Expiration Date:
##########################
```

3. Test your function:
   - Create a test function `void listSavedTransactionsTest(void);` to test all possible scenarios, happy-case, and worst-case scenarios.
   - Print all results of your test cases on the console window, and use the following as a guide:

```
Tester Name: your name
Function Name: listSavedTransactions
Test Case 1:
Input Data:
Expected Result:
Actual Result:
Test Case 2:
Input Data:
Expected Result:
Actual Result:
.
.
.
Test Case n:
Input Data:
Expected Result:
Actual Result:
```

4. You should deliver the test function as well as a screenshot of the results on the console.

Explain your work

1. Record a video where you discuss each function you implemented in this module.
2. Explain and execute all test functions you made.
3. The video is **4 minutes maximum**.
4. You may record it in Arabic or English.
5. Muted videos will not be acceptable.
6. **All of the above are mandatory to pass this criterion**.

Implement the application

CRITERIA	MEETS SPECIFICATIONS
Fill in application.h file with functions' prototypes	1. Use the following prototypes as-is: `void appStart(void);` 2. You should deliver a screenshot for your application.h file.
Implement appStart function	1. Please refer to the **flow chart** attached under the **instructions video** in order to implement this implementation. 2. You should deliver **all project folders and files**.
Explain your work	1. Record a video where you discuss each function you implemented in this module. 2. The video is **2 minutes maximum**. 3. You may record it in Arabic or English. 4. Muted videos will not be acceptable. 5. **All of the above are mandatory to pass this criterion**.

Testing the application

CRITERIA	MEETS SPECIFICATIONS
Transaction approved user story	1. As a bank customer have an account and has a **valid** and **not expired card**, I want to withdraw an **amount** of money **less than the maximum allowed** and **less than or equal to the amount in my balance**, so that I am expecting that the transaction is **approved** and my **account balance is reduced by the withdrawn amount**. 2. You should deliver a screenshot of the test result on the console.
Exceed the maximum amount user story	1. As a bank customer have an account, that has a **valid** and **not expired card**. I want to withdraw an amount of money that **exceeds the maximum allowed amount** so that I am expecting the transaction **declined**. 2. You should deliver a screenshot of the test result on the console.
Insufficient fund user story	1. As a bank customer have an account and has a **valid and not expired card**, I want to withdraw an amount of money **less than the maximum allowed and larger than the amount in my balance** so that I am expecting that the transaction **declined**. 2. You should deliver a screenshot of the test result on the console.
Expired card user story	1. As a bank customer have an account and a **valid but expired card**, I want to withdraw an amount of money so that I expect that the transaction **declined**. 2. You should deliver a screenshot of the test result on the console.
Stolen card user story	1. As a bank customer have an account and has a **valid and not expired but stolen card**, I want to block anyone from using my card so that I am expecting that any transaction made by this card is **declined**. 2. You should deliver a screenshot of the test result on the console.
Explain your work	1. Record a video where you discuss each test case. 2. The video is **4 minutes maximum**. 3. You may record it in Arabic or English. 4. Muted videos will not be acceptable. 5. **All of the above are mandatory to pass this criterion**.

Suggestions to Make Your Project Stand Out!

1. In getCardPAN function:
   - Provide the PAN as a Luhn number.
2. Implement all optional functions.
3. For the server-side accounts DB:
   - Instead of a global array create a text file "Accounts DB.txt" that stores all account data and read this file into your application.
4. For server-side transactions DB:
   - Instead of a global array create a text file "Transactions DB.txt" where you will save all transactions and read if you need.
5. Test your application against the Fraud card:
   - As a bank administrator, I want to issue my own cards, so I am expecting that any transaction made by any fraud card (failed in Luhun check) is declined.