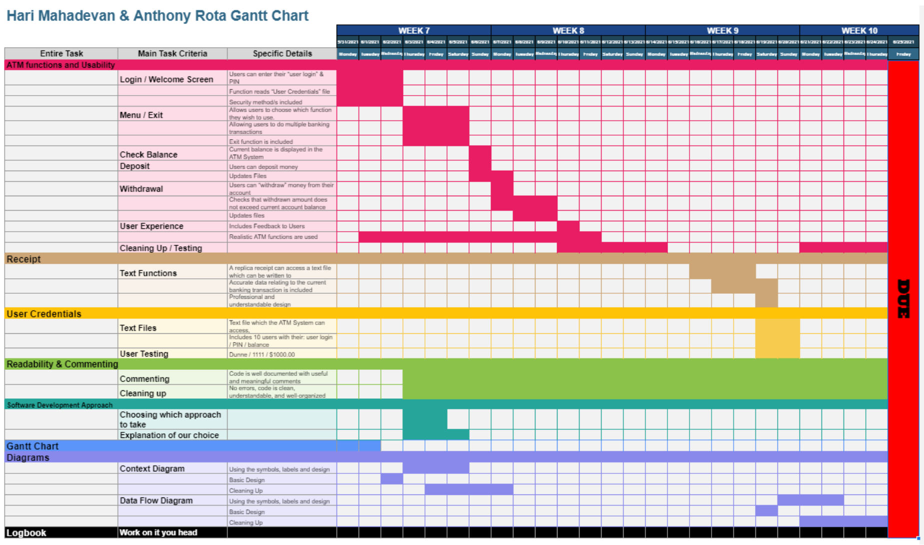
Date	Work Completed	References / Links
31/05/2021	Hari: Has started the gantt chart and put most of the things, just have to organise whenever and put all of the specific work to do and colour it in. He's finished it.	Commits: feat: login fix: db dump json spacing fix: typo pin not password References: https://github.com/anthonyrota/school-yr11-learning-text-tool https://stackoverflow.com/questions/9594125/salt-and-hash-a-password-in-python/56915300#56915300 https://stackoverflow.com/questions/23164058/how-to-encode-text-to-base64-in-python



Anthony: Looked at previous assignment for inspiration and prototyped login and menu screens as well as added the other screens (check balance, deposit, withdraw, view account) which are empty.

The user credentials are stored in a json file with the pin being hashed.

The credentials are replicated in an insecure txt file which is human readable (contains a table of credentials with pins being unhashed) for the purpose of making the credentials accessible to the marker

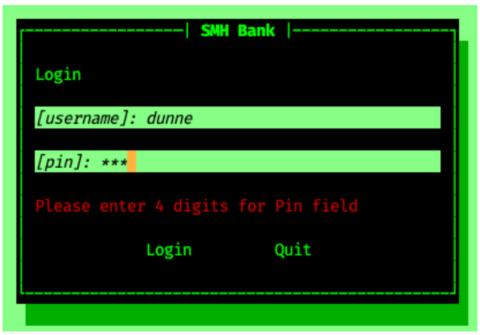
Challenge: preventing parallel access to db file in concurrent executions of the program, which will result in conflict

Current DB file example:

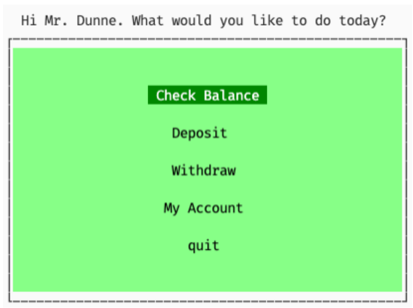
{"accounts":{"bd65600d-8669-4903-8a14-af88203add38":{"username":"dunne","name":"Mr. Dunne","pin":["/kP61/Z97HSGRg6TwBtRhw==","lGcchS3HR5CJjlwF0Sv1WTxdy1SAtxAQfzpkAT452MM="],"balance":100000}},"account_username_to_account_id":{"dunne":"bd65600d-8669-4903-8a14-af88203add38"}}

Current unsecure user credentials txt file example:

```
{"bd65600d-8669-4903-8a14-af88203add38":1111}

IF YOU ARE NOT MR. DUNNE CLOSE THIS FILE IMMEDIATELY😡😡😡😡😡!!!!!!!!! YOU ARE
INTELLECTUALLY TRESPASSING!!1!!1!
+--------------------------------------+----------+-----------+------+----------+
|              account_id              | username |    name   | pin  | balance  |
+======================================+==========+===========+======+==========+
| bd65600d-8669-4903-8a14-af88203add38 | dunne    | Mr. Dunne | 1111 | $1000.00 |
+--------------------------------------+----------+-----------+------+----------+
```



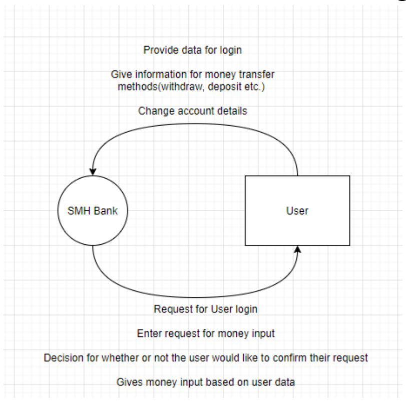
```
[username]: dunne
[pin]: ***
Please enter 4 digits for Pin field
```



Hi Mr. Dunne. What would you like to do today?

02/06/2021

Hari: Has done the context diagram, can still edit it


Provide data for login
Give information for money transfer methods(withdraw, deposit etc.)
Change account details
SMH Bank
User
Request for User login
Enter request for money input
Decision for whether or not the user would like to confirm their request
Gives money input based on user data

Anthony: Redesigned style (Changed colors and added angel brackets to buttons).

Fixed username validation. Before, usernames could be any non-empty string. Now, usernames must start with a letter (a-z) and can only contain letters and numbers.

Implemented signing up for a new account.

- The login screen has a Sign Up button which takes the user to a sign up screen
- The Sign Up screen contains a dialog which has a Username input, a Name (full name) input, and a Pin input which are the details of the account to be made
- The user can submit the form by clicking the “Sign Up” button, or can go back to the login screen by clicking the “Cancel” button
- The username is checked to be unique, and if it is not, an error message is shown to the user that the username is already taken
- If there are no errors and the user submits the form, then a new account is created and inserted into the database (by adding the details to the json’s “accounts” property, and adding the username to account id mapping in the json’s “account_username_to_account_id” property, and adding the 4-digit pin to the unsafe txt db, which is used for testing and so the marker can see all the unencrypted details in a nice table report.)
- When signing up, the pin is encrypted through the hash_new_password function, which returns a salt and pw_hash to be stored in the database in the account’s “pin” field as a fixed length array containing [salt, pw_hash]
- Once signed up and once the new user account is inserted into the database, the user is automatically logged in to the account (by setting the session’s account_id to the randomly generated account_id of the new account), and then the user is taken to the menu screen
- The new account has $0 in it

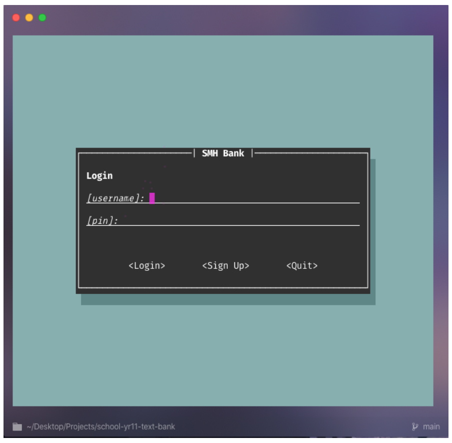
Commits:
feat: implement most things

fix: decorator spacing

References:
https://stackoverflow.com/questions/410221/natural-relative-days-in-python

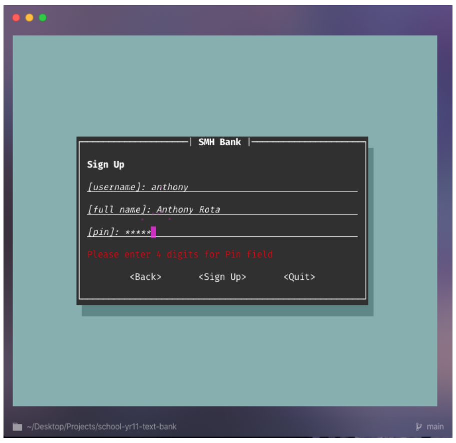
https://stackoverflow.com/questions/13890935/does-pythons-time-time-return-the-local-or-utc-timestamp

https://stackoverflow.com/questions/11743019/convert-python-datetime-to-epoch-with-strftime



```
| SMH Bank |
Login
[username]:
[pin]:

<Login>    <Sign Up>    <Quit>
```

~/Desktop/Projects/school-yr11-text-bank   main

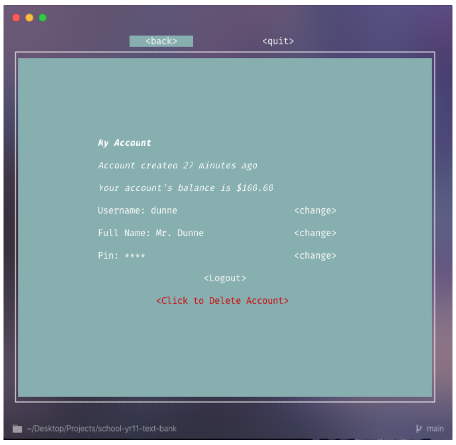


```
| SMH Bank |
Sign Up
[username]: anthony
[full name]: Anthony Rota
[pin]: *****
Please enter 4 digits for Pin field
<Back>    <Sign Up>    <Quit>
```

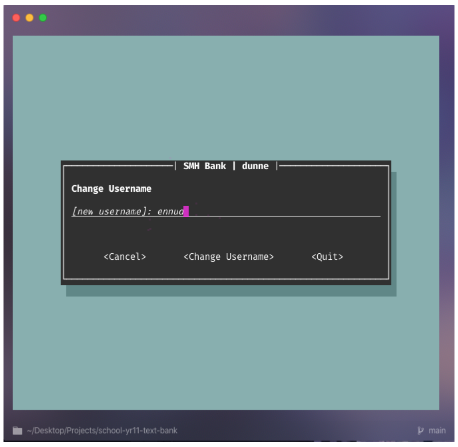
~/Desktop/Projects/school-yr11-text-bank   main

Added account creation date to database
- This is done by calling datetime.now().timestamp() to save a utc timestamp (timezone independent), which is then saved in the account in the database under the “created_at” key
- When the account is read from the database, this timestamp is parsed into a local date through datetime.fromtimestamp(account['created_at'])

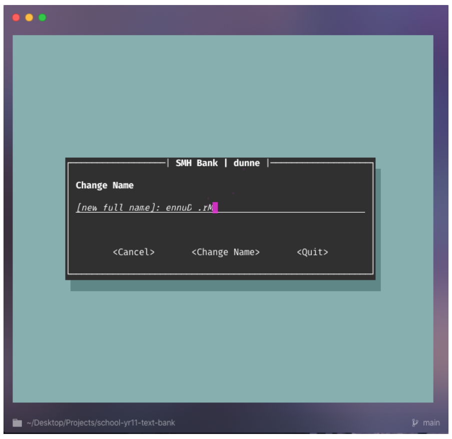
Implemented My Account screen
- This screen shows the user all the details associated with their account, and allows them to change their details
- The account creation date is shown at the top. If the creation date was within 24 hours then it will be shown as a relative time (a few seconds ago, an hour ago, etc.). Otherwise, the creation date is shown as a full date (ie. 12/11/2021) in dd/mm/yyyy format
- The user’s current balance shown on this page, formatted as dollars
- The user’s username, password and pin are also shown
- Next to each piece of data (username, name and pin) there is a button to change the piece of data which hooks to a dialog screen where the user can enter the new value and update it. This input is validated and the new value is stored in the database once the dialog is submitted.
- The user can logout from this screen, which takes them back to the login screen to login using their account or another account
- The user can finally delete their account from this screen. Clicking to delete their account takes the user to a “Delete Account” dialogue where they confirm they want to delete their account. This takes them to a second confirmation dialogue where they have to enter their username. This input is validated, that the username entered is equal to their account’s username (case insensitive check). If so, they can click the red button to delete their account, which takes them back to the login screen, and deleted the account from the database



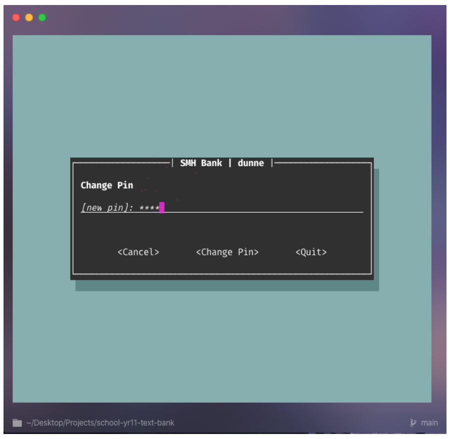
```
<back>          <quit>

My Account
Account created 27 minutes ago
Your account's balance is $166.66
Username: dunne                  <change>
Full Name: Mr. Dunne             <change>
Pin: ****                        <change>
            <Logout>
      <Click to Delete Account>

~/Desktop/Projects/school-yr11-text-bank                    main
```



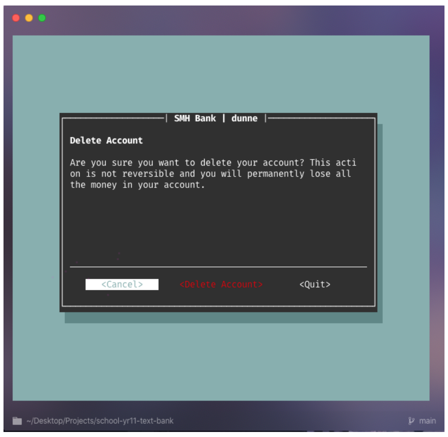
```
──────────| SMH Bank | dunne |──────────
Change Username
[new username]: ennud

   <Cancel>    <Change Username>    <Quit>

~/Desktop/Projects/school-yr11-text-bank                    main
```



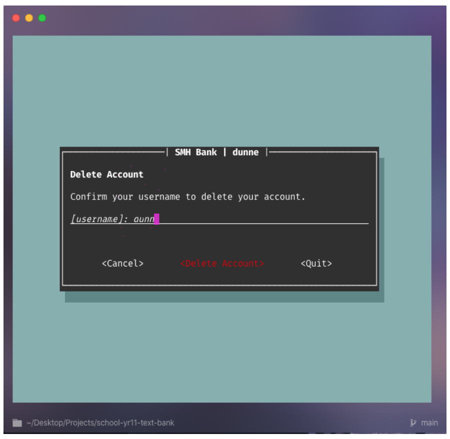
```
──────────| SMH Bank | dunne |──────────
Change Name
[new full name]: ennuD .rM

   <Cancel>    <Change Name>    <Quit>

~/Desktop/Projects/school-yr11-text-bank                    main
```



```
──────────| SMH Bank | dunne |──────────
Change Pin
[new pin]: ****

   <Cancel>    <Change Pin>    <Quit>

~/Desktop/Projects/school-yr11-text-bank                    main
```



```
──────────| SMH Bank | dunne |──────────
Delete Account
Are you sure you want to delete your account? This acti
on is not reversible and you will permanently lose all
the money in your account.

   <Cancel>    <Delete Account>    <Quit>

~/Desktop/Projects/school-yr11-text-bank                    main
```



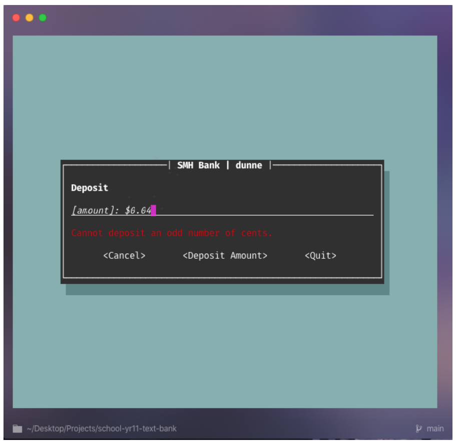
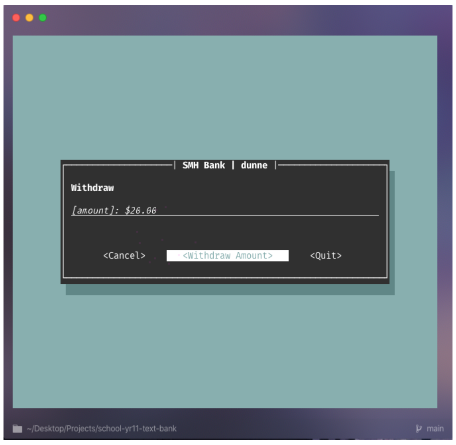
```
──────────| SMH Bank | dunne |──────────
Delete Account
Confirm your username to delete your account.
[username]: dunn

   <Cancel>    <Delete Account>    <Quit>

~/Desktop/Projects/school-yr11-text-bank                    main
```

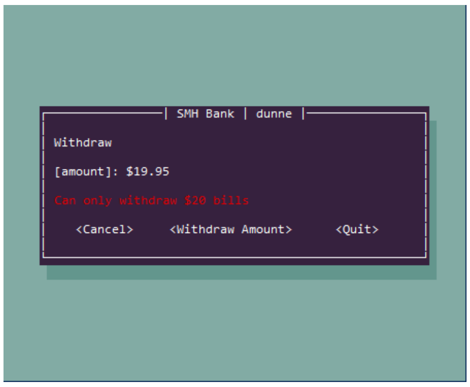
Implemented deposit/withdraw screens.

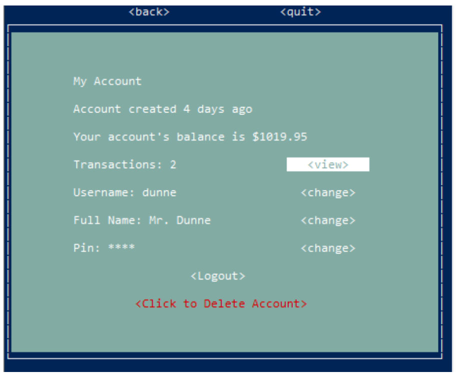
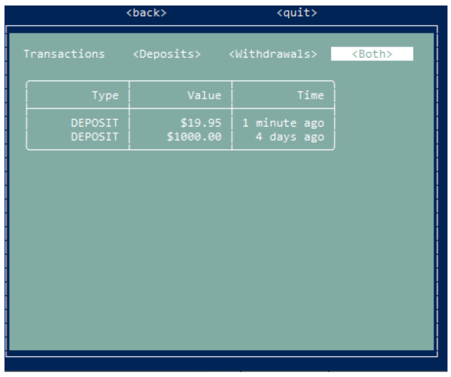
- These screens are dialogues with a singular input where the user can input the value they want to deposit/withdraw in dollars
- The input is initially validated with the regexp `^(\d+(\.\d{0,2})?|\.\d{1,2})$`
  - The input can be 0.01, 0.1, 1.0, 0., .0, 1., .1, etc but not a singular "." There can only be a maximum of 2 decimal places otherwise the input is invalid.
- The amount entered is then parsed as cents.
- If the amount entered is $0 then it is not accepted as it does not make sense to deposit/withdraw $0
- The amount of cents entered is checked to be a multiple of 5 cents to ensure that it is realistic to Australian currency

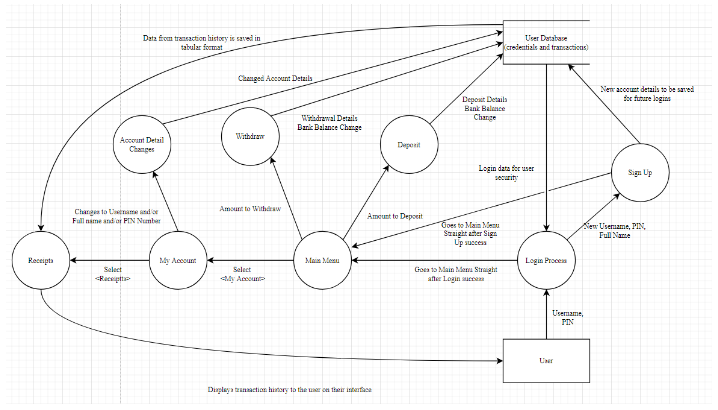
Depositing

- If the amount is valid then when the user submits the form, the amount entered is added to the user's balance in the database which is then saved, and the user is taken back to the menu screen

	Withdrawing
	- The amount entered is checked to be less than or equal to the account's balance. If the amount entered is greater than the account's balance, an "Insufficient funds" error message is displayed
	- If the amount is valid then when the user submits the form, the amount entered is subtracted from the user's balance in the database which is then saved, and the user is taken back to the menu screen
	
	
	`SMH Bank | dunne`
	`Deposit`
	`[amount]: $0.04`
	`Cannot deposit an odd number of cents.`
	`<Cancel>   <Deposit Amount>   <Quit>`
	`~/Desktop/Projects/school-yr11-text-bank   main`
	
	
	`SMH Bank | dunne`
	`Withdraw`
	`[amount]: $20.00`
	`<Cancel>   <Withdraw Amount>   <Quit>`
	`~/Desktop/Projects/school-yr11-text-bank   main`
	
	CHALLENGE: store transactions and save receipts	

03/06/2021	<u>Hari</u>: Anthony explained what some of the code means so I have an understanding as to how he is creating this project and what the variables and functions specifically do and how they interact with one another. We have decided that for me to do an equal amount, I will comment on all of the code so that Mr Dunne can see that I do understand what is happening and also implement the receipts. During this lesson, I will show how we are using an
	
	<u>Anthony</u>: I realised our ATM is not realistic as most ATMs do not allow for the withdrawal of coins and small amounts of cash (for example, a real ATM may only allow withdrawing $20/$50 notes). This is different to our current implementation as we allow for the withdrawal of any multiple of 5 cents. Started fixing the code.
	Explained the code to Hari.
	
	Fixed the ATM withdrawal process such that the ATM only "dispenses" $20 bills, and thus the user can only withdraw a multiple of $20. I implemented this by checking that the amount of cents requested to withdraw is a multiple of 2000 (that is, that the amount requested to withdraw is a multiple of $20).
	
	
	`SMH Bank | dunne`
	`Withdraw`
	`[amount]: $19.95`
	`Can only withdraw $20 bills`
	`<Cancel>   <Withdraw Amount>   <Quit>`
		<u>Commits</u>:
		fix: can only withdraw $20 notes only

	Also removed trailing fullstops in some of the text fields.	
04/06/2021	Hari: I have completed the software development approach explanation and met all of the requirements set for it.	
05/06/2021	Anthony: Implemented the recording of all monetary transactions associated with an account. The transactions are recorded in two categories: deposits and withdrawals. These are recorded in the database under each account's "transactions" field, which is an array of transaction objects. Each transaction object has a type, which when serialized is either "d" for deposit or "w" for withdrawal. I implemented a method on the database class "record_account_transaction", which takes an account_id, the transaction type, and the transaction value (in cents), and inserts it into the database by appending it to the account's "transactions" array. The serialized transaction object inserted into the database holds the type of transaction, the value of the transaction, and the timestamp of the transaction. When the user submits the deposit/withdrawal forms, the database's "record_account_transaction" method is called with the transaction type "d"/"w" respectively as well as the number of cents deposited/withdrawn. When fetching the account details in the "get_account_from_account_id" method, the values in the transaction objects in the transactions array in the database are deserialized. The total number of transactions made by an account is now shown in the account screen. There is also now a button which the user can click that takes them to a transactions screen which lists their account's transaction history. This transaction history page shows a table which lists the account's transactions in reverse chronological order. The columns of this table are firstly the type of transaction (deposit/withdrawal), the amount deposited/withdrawn, and the timestamp of the transaction. These values are first accumulated into a two dimensional list of rows of cells which is then converted to text using the tableprint library. I also implemented a filter on this screen where the user can filter the table to only show deposits, to only show withdrawals, or to show both.  My Account Account created 4 days ago Your account's balance is $1019.95 Transactions: 2 Username: dunne Full Name: Mr. Dunne Pin: ****  Transactions Type | Value | Time DEPOSIT | $19.95 | 1 minute ago DEPOSIT | $1000.00 | 4 days ago Implemented saving receipts of each transaction. These receipts are saved in the "<projectPath>/receipts/{username}" folder. This folder is created in the database's "make_account_with_details" method. When the user changes their username, this folder is renamed to the new username at the end of the "change_account_username" method. When the user deletes their account, this folder is deleted at the end of the "delete_account" method. In the database's "record_account_transaction" method, the date, time, type of transaction (deposit/withdrawal), amount deposited/withdrawn, and the updated account balance is formatted into a textual table using the tableprint library, and is combined with the header "SMH BANK", the account's name, and "APPROVED", then is written to disk at	Commits: feat: record transactions feat: receipts References: https://stackoverflow.com/questions/303200/how-do-i-remove-delete-a-folder-that-is-not-empty https://github.com/nirum/tableprint

	"<projectPath>/receipts/{username}/{timestamp}.txt".	
07/06/2021	Hari: Fixed text encoding (error on windows school computed when testing). Now explicitly "utf-8" when opening a text file. Began Structured Walkthrough on google slides. Anthony: When deleting an account, the username entered had to be lowercase to be checked as valid. As the username is stored in lower case in the database, this can be confusing for the user. For example, the user may have submitted "Dunne" as their username, and so would expect that typing "Dunne" into the delete account username confirmation form would be valid, which currently it wouldn't as the user would have to have typed "dunne". Now the check is case sensitive. This means that if the user enters "Dunne" or "DUNNE" or "dUnNe" into the delete account username confirmation form, it will be accepted as valid.	Commits: fix: text encoding and deleting account username validation References: https://stackoverflow.com/questions/27092833/unicodeencodeerror-charmap-codec-cant-encode-characters
10/06/2021	Hari: Completed the Structured Walkthrough. Did the Level 1 Data Flow diagram but I need to double check it with Anthony and Mr. Dunne tomorrow and fix up any mistakes that do not correlate with the functions of the code.  User Database (credentials and transactions) Data from transaction history is saved in tabular format Changed Account Details Withdrawal Details Bank Balance Change Deposit Details Bank Balance Change New account details to be saved for future logins Account Detail Changes Withdraw Deposit Sign Up Login data for user security Changes to Username and/or Full name and/or PIN Number Amount to Withdraw Amount to Deposit Goes to Main Menu Straight after Sign Up success New Username, PIN, Full Name Receipts My Account Main Menu Login Process Select <Receiptts> Select <My Account> Goes to Main Menu Straight after Login success Username, PIN User Displays transaction history to the user on their interface Anthony: Edited Structured Walkthrough. Also submitted assignment backup. Used the tendo library to lock the program to prevent concurrent execution. This means that if an instance of the ATM program is already executing, spinning up another instance will do nothing and will exit with the error message "ATM is already running..." This solves the issue of database locking/versioning, preventing database conflict as now the program cannot be executed more than once at a time.	Commits: feat: prevent concurrent execution References: https://tendo.readthedocs.io/en/latest/#module-tendo.singleton
11/06/2021	Anthony: Edited Software Development Approach	
12/06/2021	Hari: Edited the Structured Walkthrough to make things clearer. Completed the Level 2 Data Flow Diagram but I have to get it checked with Anthony at school. Anthony: Edited Structured Walkthrough	References: https://github.com/prompt-toolkit/pymux
17/06/2021	Anthony: Started adding more comments to make the code clearer.	
18/06/2021	Hari: Edited Data Flow diagrams to make things better. Anthony: Testing code for bugs, didn't find any and did some comments	
23/06/2021	Hari/Anthony: Finishing up the assignment, the only thing we worked on this lesson was commenting on the code and making it more	

	beautiful.	