Aidan Lawson & Derek Liu

26/04/2019

Ruby Terminal Application Assignment 1

Banking Application

Ruby Application Design Documentation

Application Ideas:

* Banking App
* Expert System
* Recipes suggestion
* Phone Operating System
* To Do App
* Health and Fitness App
* Note Taking App
* Custom Itinerary App

# Application Feasibility Assessment

* Banking App
  + Easy to present and straight forward design
  + Easy to Implement DRY code
  + Easy to make object oriented
* Expert System
  + Scope too large. The amount of user cases that would have to be taken into consideration when creating the app would be far too large given the time frame.
  + The App would be very text heavy and would require the user to answer a large number of questions in order to give a diagnosis, which can make for a bad user experience
* Recipes App
  + Straightforward design that in terms of being able to make the code DRY and Object Oriented
  + However once again the scope is also very large. Given the number of ingredients that are available coding in all the possible ingredients would be time consuming and overwhelming when taking into account the time frame allocated to the assignment. Furthermore, when taking into account all the possible recipes it increases the scope even further. The app being coded in terminal would also be very text heavy and wouldn’t make for the greatest user experience
* Phone Operating System
  + Straightforward design that in terms of being able to make the code DRY and Object Oriented.
  + Very Text heavy and given the lack of presentable UI it wouldn’t make for a good user interface.
  + Operating System would be very simple and basic
* To Do App
  + Difficult to present the app in an aesthetic manner when only using the terminal to make the app
* Tax App
  + Not enough content to be a standalone application, thus it was decided to make the Tax App a feature of the banking app
* Custom Itinerary Planner
  + Difficult to present the app in an aesthetic manner when only using the terminal to make the app
  + Too much text to be presented in the app without an appealing UI would make for a bad user experience
* Fitness App
  + Once again would be a very text heavy app without the use of an external UI, impairing the user experience.
  + Too large of a scope to cover with the amount of time given.
* Health App
  + Falls into the same category as the Fitness app where user experience would suffer as a result of an overload of text to the user.
* Note Taking App
  + Difficult to present the app in an aesthetic manner when only using the terminal to make the app
  + Large scope that would be hard to achieve in the given time. Also quality of the app suffers due to only being executed in the terminal
* Chosen Application:
  + Banking App. The banking application was chosen out of all the options as it was seen by all members to be the most feasible project, while still allowing for the app to be coded DRY and Object Oriented. The app was also applicable to the real world and was achievable in the given time, and was believed by all team members that a high quality project would be delivered at the conclusion of the assessment

Banking Application Planning

Project Planning Tools

* Trello
  + In order to keep on top of the Project a Trello board was created to ensure that each member of the project had tasks delegated to them and knew the deadline for each assigned task. When a task was created, a deadline and team member was assigned to the specific task, the individual team member assigned to the task would be responsible for the management of that task on Trello; updating the status of the task and adding any necessary comments or descriptions and ensuring that the task is completed before the deadline.
* Github
  + Github was the chosen platform for application version control. Github was chosen due to the fact that it would allow for the code to be shared between each team member. Any changes made to the code would be documented by the specific team member during the committing stage. This allowed team members to constantly be aware of any major changes to the code and keep up to date with the latest revisions of the project.
  + Link to GitHub repository: <https://github.com/liuderek97/Ruby-Application-Assignment-1>
* Discord
  + Discord was the main communication platform employed by all members when discussing the project outside of contact hours. Discord was the chosen platform for communication as all users were familiar with the platform and using a communication platform that all users were comfortable with would allow for unhindered communication channels, and allow effective and efficient communication.
* Collalbedit
  + Collabedit is a code sharing platform allowing for multiple users to edit the same console online. Team members would use Collabedit when working on the application to allow for effective collaborative teamwork. Collabedit allowed for members to work on the same program in real time, and was essential to the completion of the application, as it allowed for members to discuss and plan the structure of the code when designing and implementing features present in the application. Furthermore it allowed for members to not only understand the thought process but the logical flow of certain functions as the code could be explained and produced in real time allowing for members to discuss their thought process, and inner workings for the various functions and logical flow of the application.

Application Planning Stage and Feature Explanation

After completing a feasibility assessment on all the proposed applications, and selecting the application after the evaluation, the team started planning out the features and structure of the code. In order to plan the application, members sat down and wrote out what features the application would have, how they would be implemented and propositions of features that could be added on during the completion of the project. In the early stages of the project a majority of the early pseudocode, structure and ideas were written down in a book discussed between each member. After writing down the early tasks that needed to be completed, tasks were then created and uploaded to a Trello board. When the tasks were moved to the Trello board they were then assigned to a team member and set a deadline, it was then up to each individual member to then handle all requirements in relation to each task; completing the task, adding any necessary comments about the task, ensuring the task be completed and delivered and moving the task to the appropriate list: To Do, Doing and Done. After early planning stages the following features were proposed and implemented into the application. The proposed and implemented features were as follows.

* Banking Application features:

1. Main page where user is presented with multiple options.
2. Application stores user account information.
3. Users will be able to login to their accounts to check their account details and balance.
4. Users will be allowed to deposit and withdraw money into their accounts.
5. First time users will be able to create an account
6. Users will be able to have several accounts under the same name, as long as they are different types of accounts e.g. Savings and investment accounts
7. Each account will have a unique security identifier to help with account actions and account verification.

* Application feature 1: Storing user account information
  + The application stores two types of data; User data and account data. When starting up the application, users will have the option to login if they have already are a user and have an account, or will be able to create an account if they are a first time user.
* Application feature 2:
  + The application will be able to store not only user information but account information. Both user information and account information will be stored in a dictionary. It is imperative that the information is not only stored, but stored correctly, so that when a user uses the application its stores the balance of each account and the details of each user. Users will be able to check their personal details in relation to their accounts such as name of the account, account balance, the unique id assigned to each account and the name of the account. These pieces of information will be also be essential to perform any account actions such as creating an account or user, or withdrawing and depositing from an account.
* Application feature 3:
  + By allowing users to login to their account, users will be able to see information that is only available to them. It is important that a login system is implemented and that users will be able to login for the protection of each users data. It would not only be unethical but an breach of personal security if all account information of all users would be available to all users. Users will only be logged in if they enter their own personal usernames and passwords created by themselves for security purposes.
* Application feature 4:
  + By extension of being a banking app and being able to access account information individuals will be able to deposit and withdraw money to their accounts. The application will allow the user to deposit any amount that they wish, as well as withdraw any amount that they wish, as long as the amount they desire to withdraw does not exceed the balance of the account. After depositing or withdrawing from their accounts, individuals will see the balance of their account update immediately after the transaction has been successfully completed
* Application feature 5:
  + By allowing first time users to create an account the core functionality of the application will increase. First time users after creating an account with the bank will then be able to create a bank account to which they can with or deposit money into. This is essential as not all individuals using the app will have an existing account
* Application feature 6:
  + Users will be able to open as money accounts as they desire under the same name for different purposes. This will allow users to divide and control their bank accounts as they see fit, and was implemented to imitate real life banking systems.
* Application feature 7:
  + By having a unique identifier for each individual accounts for security purposes as well as to allow for users to hold different accounts under the same username, as it will be the differentiating factor when performing account actions. Whenever the user wants to perform an account action they will have to input the unique of the account that they’d like to withdraw and deposit from to ensure that they are withdrawing or depositing to the correct account.

Application Code structure

During the planning phase of the project it was proposed that the application be split into separate files. The application was split into separate files in order to create cleaner code allow for the application to be coded with a logical flow. The application was split into separate files, being:

* bank\_app.rb
* atm.rb
* account.rb
* accountdb.rb
* user.rb
* userdb.rb.
* bank\_app.rb is the file that calls the ATM function and starts the app.
* atm.rb is the file that contains the main user interface in which the user will be interacting with while using the applications. The atm file contains functions that allow the user to login or create an account and once logged will allow the user to make deposits or withdrawals. In the case where the user is not already an existing user, the atm file will allow for the user to create their user account then open a bank account. The functions in the file are:
  + run
  + login
  + create\_user
  + open\_account
  + deposit
  + withdraw
  + menu
* **run:**
* **login:**
* **create\_user:**
* **open\_account:**
* **deposit:**
* **withdraw:**
* **menu:**
* account.rb is a separate file containing a class that is responsible for creating a class object of type Account. The Account object holds 4 types of unique information:
* **Unique Id:** The unique id of the user, designed to identify different user accounts and allow for easier calling of account methods when dealing with certain functions such as the withdraw and deposit function. The Unique id was also implemented to allow for one user to hold several accounts. A user was able to create several types of accounts under the same username, however the unique id would allow for the user to choose which account they would like to perform any actions with.
* **Username:** The account was assigned a username which was created by the user when prompted to enter a user name which they desired. The username was utilised when logging in an existing user, as it allowed for the application to check the existence of the account using the username when logging in a user with an existing account. It also allowed for different users to hold several types of accounts under the same username.
* **Account Name:** The account name was a field used to specify the type of account that each user had. Users were able to have different types of accounts under the same username. The account name along with the unique account id would be essential in specifying and differentiating between the different accounts that the user held. For example a user would be able to hold two accounts under different accounts names such as a “savings” account and an “investment” account
* **Account Balance:** The account balance is responsible for storing the monetary value and balance of the account. When a new account is created the account will have a balance of $0. The user is then free to change the value of the balance via depositing or withdrawing from the account. The account balance is updated and changed after each account action.

After the account object is created the object is passed through the save method in the account class to allow the object to be stored into an array of hashes. The save method calls a method in accountdb.rb which is responsible for the saving of the account into the dictionary.

* Accountdb.rb is the module that is responsible for most of the actions that take place when dealing with accounts objects. The account.rb module has the following methods:
  + **find\_by\_username(username):** The find by username method was the main method user in the application when it came to verifying if an account existed when the user would be logging in. The method worked by requiring the user to enter their username which would then be passed to the method as an argument. After the username was passed to the method, the method would then search through the dictionary to find any matching usernames. If the username existed in the dictionary the evaluating expression would evaluate to true and then create call the method make\_instance to create an instance using the username passed and all other information associated with the username. The expression used when evaluating the existence of the username if evaluated to be true would then allow the user to login with that username and complete account actions such as withdraw and deposit for accounts belonging to and associated with the username. When the expression evaluates to false however the user is then made aware that the entered username could not be found or does not exist and is then given the option to either attempt to re-enter their username or create an account if they don’t have one
  + **find\_by\_id(id):** The find\_by\_id function much like the find\_by\_user\_name function was used to check the existence of the account. However the situations in which it was used differed from its counterpart. Whereas find\_by\_username would be essential to the logging in function of the application the find\_by\_username function would be called and used whenever the user would be performing account actions, specifically withdrawing and depositing to accounts. As mentioned before each account had a unique account id which would assist in identify accounts held by different user, or different accounts held by the one user. When performing a deposit or withdrawal and after the user had been logged in, the user would be required to enter their unique account id. After entering the unique id of the account that they would like to peform the action on the find\_by\_id function would be called and the account id would then be passed as an integer to the function as an argument. Once the argument was passed to the function would then search through the dictionary to find any matching id’s. If the id existed in the dictionary the evaluating expression would evaluate to true and then create call the method make\_instance to create an instance using the id passed and all other information associated with the id.
  + When the expression evaluates to false however the user is then made aware that the entered id could not be found or does not exist and is then given the option to either attempt to re-enter their id to continue their deposit or withdrawal.
  + **make\_instance(account\_hash):** The make instance function is responsible for instantiating an abject from a hash. The method is called in the functions find\_by\_username and find\_by\_id. The purpose of the function is to create an instance of each individual account. By creating an instance it makes the handling and updating of accounts details much more efficient as calling methods in relation to an instance allows for a more customised approach to handling the user account
  + **create** (**user\_name, name, balance):** The function create is responsible for the creation of the account. When creating the account the user will be prompted to input their username and name of the account that they would like to create and use. The balance is set to 0 when created and the unique id is created in the function by calling the new\_account\_number which creates a new unique id and assigns it to a variable which will then be used to assign a value to the id key in the hash. When calling the account function the passed arguments will be be used to create a hash with each value being assigned to a key. After being saved as a hash the hash containing all the entered account information will then be pushed onto the existing array of hashes, and will then be made an instance.
  + **save\_object\_as\_hash\_in\_array** **(account):** save\_object\_as\_hash\_in\_array is the main function that is called whenever there is a change in the details of the account. The change will mainly occur in the deposit and withdrawal method and will allow for any changes made to balance to be reflected and saved for the duration of the applications runtime. The function works by passing an account instance to the method. Once the account instance is passed the method will iterate though the array of hashes and update the values that had been changed for the specific account where the action had taken place
  + **new\_account\_number:** new\_account\_number is a relatively small and simple function that is responsible for the creation of the account id.
* user.rb is a file that contains the class User. The file is responsible for the creation of User objects and contains only one method which initializes and creates the user object. The initialize method will take the parameters: user\_name, name, age and password. After each parameter is passed the object is created.
* userdb.rb is another file that contains the module UserDB. The userdb module is responsible for most of the actions responsible for modifying and storing user objects. The UserDB module contains the following methods:
  + find\_by\_user\_name(user\_name)
  + make\_instance(user\_hash)
  + create(user\_name, name, age, password)
* **find\_by\_user\_name(username)**: The find\_by\_user\_name method works very similarly to the find\_by\_user\_name found in the accountdb.rb file in that it evaluates whether a given username exists in the user dictionary. If the username existed in the dictionary the evaluating expression would evaluate to true and then create call the method make\_instance to create an instance using the username passed and all other information associated with the username. The expression used when evaluating the existence of the username if evaluated to be true would then allow the user to login with that username and complete further actions in the application.
* **make\_instance(user):** The make instance function is responsible for instantiating an object from a user. The method is called in the functions find\_by\_username . The purpose of the function is to create an instance of each individual user. By creating an instance it makes the handling and updating of user details much more efficient as calling methods in relation to an instance allows for a more customised approach to handling the user.
* **create(user\_name, name, age, password):** The function create is responsible for the creation of the user. When creating a new user the user will be prompted to input their username, name, age and password. When calling the create function, the passed arguments will be be used to create a hash with each value being assigned to a key. After being saved as a hash the hash containing all the entered account information will then be pushed onto the existing array of hashes.