Round-Up Saver Tool

Created by: Kuran Pran Malhotra

April 2019

Problem Statement

As a college student, saving money is hard. Traditional techniques such as saving XX% of your income or depositing into a 401(k)-type plan do not work, as college students tend not to have any sort of fixed income to divide up. Therefore, the problem I hope to solve is as follows: **As a college student, I would like to create an automatic and hassle-free methodology by which I can save money incrementally to aid in long term financial planning.**

Primary User: My primary user will be myself, a college student without fixed income, and with an account at the local, on-campus Credit Union.

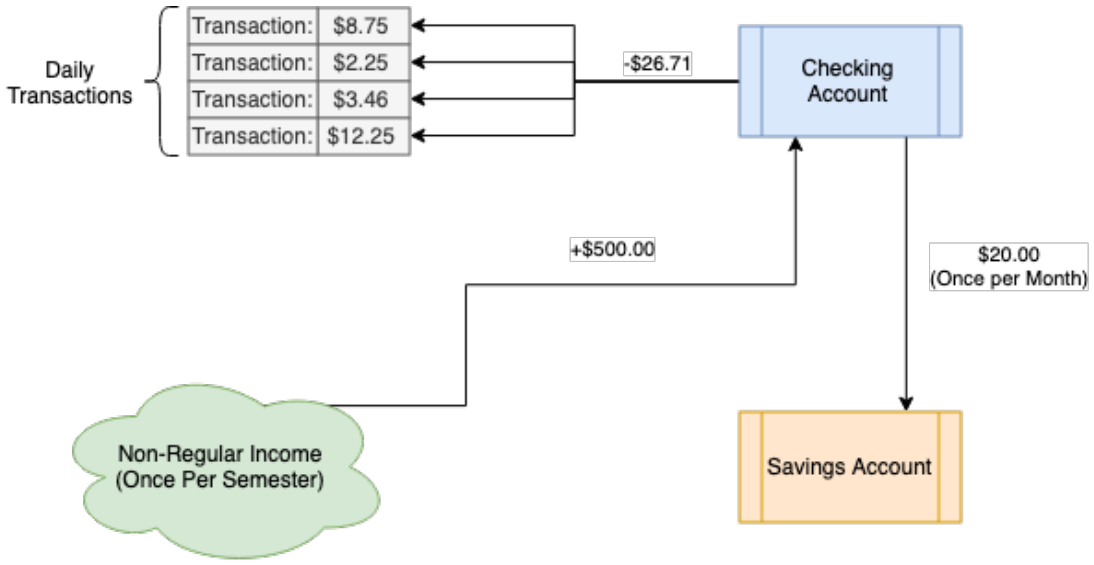
Current State Processes

One of the most common processes to address this issue is just simply transferring money directly to a savings account, and transferring money out into a checking account as needed. While this works, it mainly involves reducing a savings account on a regular basis, instead of increasing it.

Another current process is to sign up for a third party service that will save money on its own platform for you via algorithmic spending tools, and take 1% or so from those savings. This effectively negates the savings.

The most analogous tool to the solution I hope to create is to manually transfer money every so often (usually upon memory) from a checking to a savings account. The workflow of this is demonstrated below:

Figure 1: Current State Process Diagram



As the above process shows, on the average semester of 4 months, a student would save about $80.00 per semester.

Potential Solutions

One potential solution is to create a CD product where immediately of that intial $500 deposit per semester, 25% of it or a specific percentage is placed in a certificate of deposit account. This, though, does not provide flexibility on behalf of the student, and in fact penalizes the student for withdrawing funds early, should there be an emergency need.

Another potential solution is to create a tool that automatically saves the money ($20.00 per month) for you. While this does one-up the manual process that currently exists, it does not fully do justice to the problem at hand: incremental and constant savings.

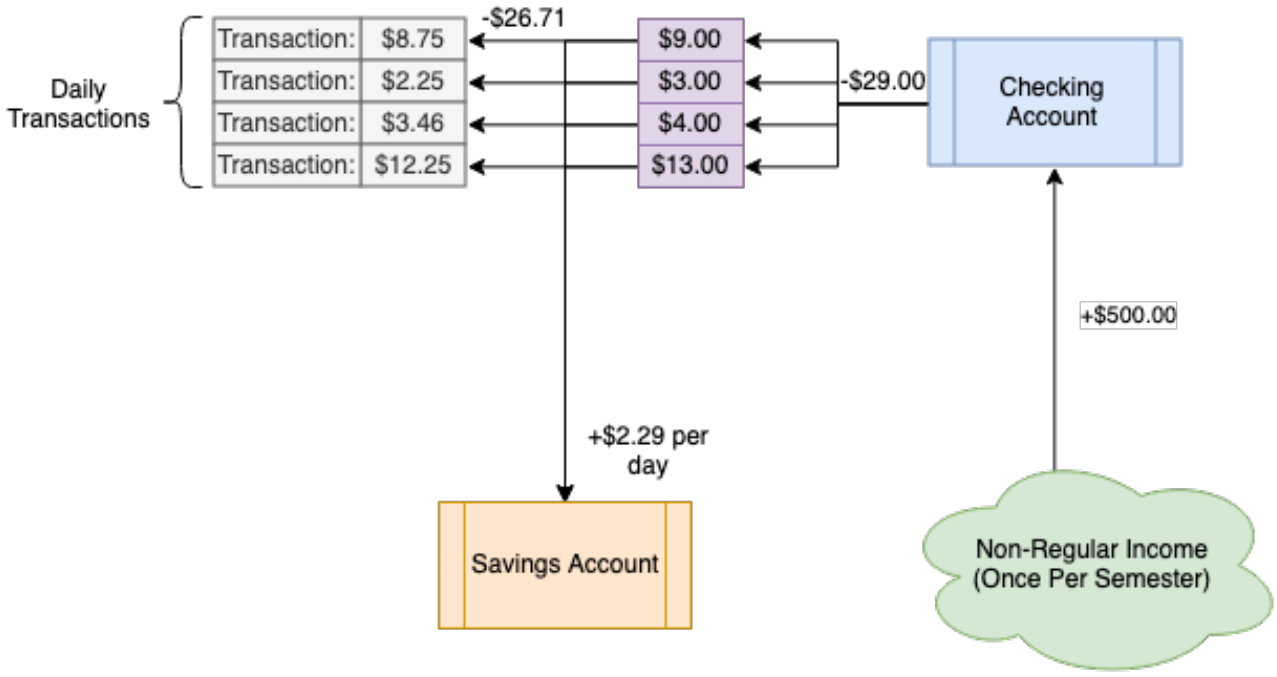
The solution I have chosen to implement not only does automate savings on an incremental and constant basis, but it is also fully free to the user, and therefore only has positives. I hope to create a tool that will automatically save money every time a debit card transaction is conducted. As these transactions occur, my system will round up the amount and transfer the difference between the transaction amount and the nearest dollar to the savings account, therefore saving money incrementally, without any noticeable difference in transaction amount on a day-to-day basis.

Proposed Solution

As mentioned, I will be creating an automatic saver tool that will round each debit card transaction up to the nearest dollar, and transfer the difference into a savings account. This will be done fully automatically, therefore requiring no work on behalf of the user after implementation, and allowing for heavy benefits with no downside.

A diagram of transaction flow is shown below:

Figure 2: Proposed State Process Diagram



As shown, this system has a minimal impact on the day-to-day transactioning of the student, and as a fully automatic solution provides much more convenience. Further, it allows for $2.29 per day in savings, amounting to $247.00 in savings per semester, as opposed to the prior $80.00. Unlike the CD options or others, should this money be needed for another purpose in an emergency situation or otherwise, it can be moved to the savings account for free by the user.

System Objectives

The system should ideally behave as follows:

- Get transactions pushed to or regularly collect transaction data from the user's bank accounts
- Create a sum total of the round-up values, and transfer that amount automatically from the checking to the savings account
- Send a daily recap of the amount saved
- Do all of the above securely, protecting the financial data of the user.

Functionality Requirements

From the user perspective, the entire process should be seamless. After the initial set-up of putting in account information, etc., there should be no user interaction with the system. The transactions should be automatically gathered from the banking platform in a secure manner. The system will be deployed to a Heroku server and will run automatically. The user should not have to upload transactions or anything of that nature.

Further, the user should not have to incur any additional cost as a result of the system. If there is not enough money in the checking account to perform a savings transfer, for example, the system should know not to overdraw the account and incur the user a fee.

In creating the round-up totals, the system should know to round up each cent-value to the nearest dollar and transfer the difference. This will ideally be done on a per transaction basis, though depending on the API functionality, it might be aggregated on a regular (hourly?) basis.

The daily recap should arrive via text message to the phone number registered with the account. It will simply state the checking balance, the savings balance, and the amount saved throughout the day.

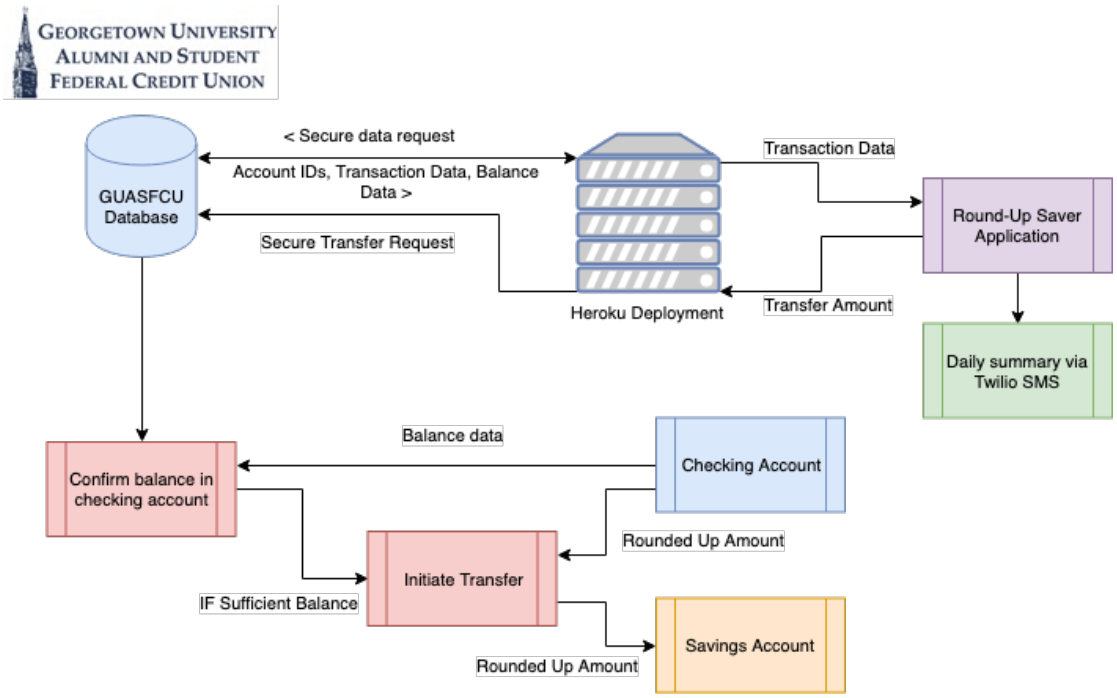
All of this must be done in a secure fashion. The encryption of the API I hope to use is SHA-256 encryption, and as such upholds the level of security I hope to incorporate into my application.

Each fundamental aspect of the project (informational, interfacing, and technological) is described in more detail below.

Information Requirements

The information required mainly is furnished from the GUASFCU database via its API. A diagram of data flow is shown below:

Figure 3: Data Flow Diagram



The data requests will be sent as API get and post requests for account data and transfer requests, respectively. Within the GUASFCU system, the request is validated, the balances and transactions are confirmed, and the transactions occur instantly and automatically. The balance, transaction, and account data are all provided as JSON responses, which will be parsed by the application. The daily summary will be sent via an API post to the Twilio platform, which will then send an SMS message to the user's cell phone.

Interface Requirements

As mentioned, the user should not have to interface with the system at all. It should be automatic and hassle-free post set-up. That set up will occur via a Heroku deployment, with the python script running every time a new transaction is posted, and once per day at the end of the day for the summary. The user will receive a text message with their daily summary of balance and savings transfers made, and that's it!

Technology Requirements

Foremost, this application will utilize the open banking API of the Georgetown University Alumni and Student Federal Credit Union (GUASFCU). This API is managed by Narmi, and their documentation will be provided within the documentation of my application.

The API allows a user to get their specified account data, transaction data, balance data, and personal profile information. It also allows the user to transact, transfer between accounts, and edit transaction meta data via a third party application.

The biggest requirement of that system is security. Each data request (Get, Post, etc.) must be encrypted with a token, secret, and SHA-256 encrypted signature in its header. There are various third party Python packages that can help create this header, one of which is HTTPS. I anticipate using one of these.

Another API I will employ is the Twilio API (and its accompanying Python package). This will be used to send the daily text message to the numer associated with the GUASFCU account.

The application will be deployed to a remote server managed by Heroku. It will run on a continuous basis as frequently as allowed by the Heroku platform, and should not require any other hardware or software to function.