Alexander Skvortsov

alexander@psu.edu
PSU ID: 943836526

Collaboration Statement:

All code was written by me. A number of 3rd party libraries were used. Those can be found in the requirements.txt file. A large number of online resources were used to learn more about the flask framework, but no code was copied.

Overview

I decided to implement Homework 2 in the form of a Flask-powered JSON RESTful API.

Instructions for running the API and the included integration tests can be found in the README.md file.

A more detailed API use documentation can be found at https://app.swaggerhub.com/apis-docs/askvortsov1/Banking-Simulation-PoC/1.0.0

Essentially, this is a proof-of-concept for an API interface to a multi-bank SaaS system. Due to the assignment requirements and the tight deadline, some significant limitations were required. More information on these limitations can be found in the README.md section.

The base of the system are banks, each of which has a number of branches. Each branch has a number of staff.

Each bank also has a number of accounts. Because accounts can have vastly different options and settings, those are configured through a separate account configuration model, which are created per bank.

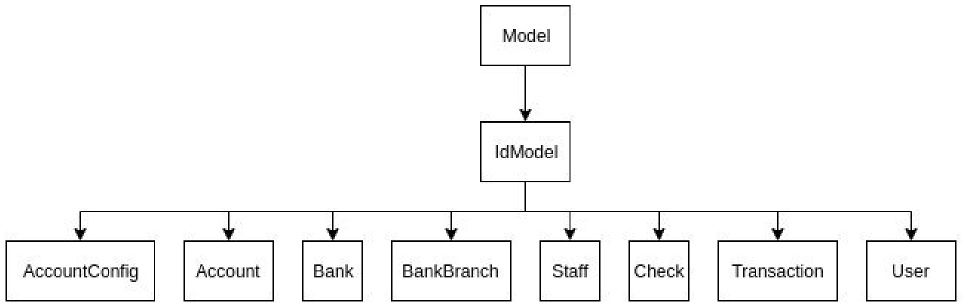
Users are created in the system independently of banks. They can have accounts or staff positions in one, none, or many banks.

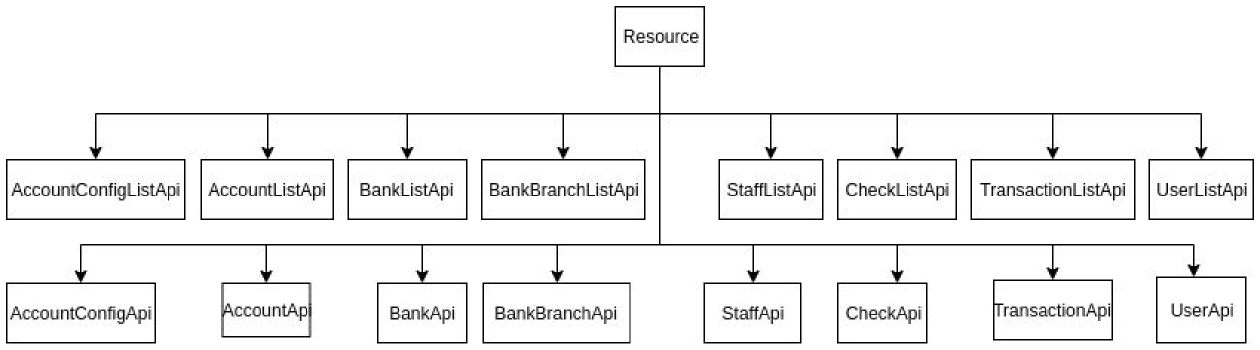
Accounts can issue checks, which can then be deposited by a different account at a later time. As in real life, checks are validated at the time of deposit.

Deposits and withdrawals are processed through a unified transaction interface. Overdrafting is supported, and deposit and withdrawal fees are applied as per the account's configuration. For savings accounts, withdrawals are limited to 6 as per U.S. Law.

Information on endpoints and parameters can be found at https://app.swaggerhub.com/apis-docs/askvortsov1/Banking-Simulation-PoC/1.0.0.

Inheritance Schema





*Model and Resource are defined in the 3rd party libraries flask-sqlalchemy and flask-restful respectively

Required Components

Property Methods:
These are used throughout the code. One example is in src/models/user.py; line 21, where a property method is used to provide access to the user's full name.

Magic Methods:
Every model implemented includes a __repr__ method. One example is in src/models/financial_instruments.py; line 67, where the representation of a check is defined to include the amount and the account issued to. In the Transaction class (src/models/financial_instruments.py; line 93), the __init__ method is used to cache the monentary amount of checks deposited in the transaction on object instantiation before calling the super() init method to instantiate the instance.

Abstraction:
The IdModel class, defined in src/base_model.py, is a base class for all models defined. It represents the abstract idea of a model, and through its own inheritance of flask_sqlalchemy.Model, provides an interface for querying models, and functionality for storing, updating, and deleting records in the database.

Encapsulation:
Encapsulation is used throughout the project. An example is in src/models/accounts; line 67, where the _balance attribute of accounts is private by convention, and is accessible through the balance property method (line 74).

In addition, the entire project is, in a way, an example of encapsulation: To the end user, the program is only accessible through the JSON API Endpoints: in this way, interface and implementation are separated, and the end user does not have access to the state of the program.

Inheritance:
Inheritance is widespread. All model classes inherit of of IdModel (indirectly through the Model attribute of the database object). An example is in src/models/accounts.py; line 54, where the Account model is defined as an inheritance of db.Model (which is IdModel).

Similarly, all API Endpoint classes inherit off of Resource. An example is src/resources/bank.py, where BankApi is defined to inherit off of Resource.

Polymorphism:
Polymorphism is similarly widespread. All API Endpoint classes inherit off of the Resource class. Support for HTTP method access is then implemented through overriding get, post, put, delete, etc. methods. Then, when executing a request, the flask-restful library calls the relevant method on Resource (which pushes it along to the subclass if implemented), or raises an exception if not implemented to let the user know that the given request method is not supported.

An example is in src/resources/bank.py; line 12 where the BankListApi endpoint defined support for the GET and PUT http methods through its get and put methods, respectively.