Coding Assessment

Background

You are building a new app to allow people the ability to assess their financial spending from other financial institutions. Your customers will have one or many accounts with different providers, e.g. ikigai, Natwest and HSBC.

Your app will be connected to all the accounts the user grants access to and will receive a webhook each time a transaction happens in one of the 3rd party platforms. On occasions, the 3rd parties resend the same webhook and there could be errors in the data they provide.

Before your team can provide the spending data to your customers, you must process this data from the 3rd parties. Part of your team will be looking at providing the API interface for the iOS device and you will be required to look at the ingestion from the 3rd parties.

Challenge

You should design and implement the initial backend logic to handle transactions from 3rd party providers so that when a webhook is received you can process this data, store it, and the team can provide this via an API interface to the iOS device.

Assumptions

1. You can ignore any code associated with creating user accounts or security.
2. Each user can have any number of 3rd party accounts but assume every user has each of the ones specified below.
3. Each provider will need to submit a transaction webhook to a particular route in your app.
4. There are 3 events as specified below:
    a. in-store-transaction
    b. online-transaction
    c. atm-withdrawal
5. The webhook should just return a 200 OK response

Requirements

1. Use standard scaffolding / cli to generate a new web application in Ruby on Rails [or MVC framework of choice].
2. Create the routing necessary to handle webhooks from each of the 2 providers below. Decide whether each provider is routed to their own controller or if you will handle all providers with the same controller

3. Create the initial migrations to handle the storage of transactions and financial institutions. Map any relevant relationships. Consider the different entities required and specification of 1-1, 1-many and many-many relationships.
4. Create the code to handle the incoming webhooks from each provider to:
    - Convert the raw webhook data into an internal transaction
    - Ensure the same webhook does not duplicate a transaction if it has already been processed
    - Store the original data from each provider
    - Demonstrate how to handle errors
    - Demonstrate knowledge of testing
    - For every transaction also store the bitcoin value (https://www.blockchain.com/api/exchange_rates_api)
    - The webhook from each provider looks like below. The events should map to our internal event types

ikigai	Bank 1	Bank 2

{
"transaction_time": "1603711958",
"id": "e0c523cd-3dfd-4206-83b4-9c0dc32dd77e",
"event": "in-store-txn",
"value": 13.98,
"status": “complete”,
"customer_id": "e0c523cd-3dfd-4206-83b4-9c0dc32dd77e"
}
 	
Events
“in-person”
“online”
“withdrawal”
	{
	"transaction_time": "2020-10-02T12-23-12",
	"id": "34523123974",
	"value": 27,
	"event_id": “27”,
	"customer_id": "e0c523cd-3dfd-4206-83b4-9c0dc32dd77e"
	}
	
	Events
	“27”
	“31”
	“2”
		{
		"transaction_time": "2020-10-02T12-23-12",
		"id": "9e04ed1e0ba19a7ff938187efc5f560a9e056d65",
		"action": “store-transaction”,
		"customer_id": "e0c523cd-3dfd-4206-83b4-9c0dc32dd77e"
		}
		
		Events
		“store-transaction”
		“ecommerce-transaction”
		“atm”

5. Document your code explaining the decisions you’ve made, clearly demonstrating why certain code is in controllers, views, models or services