Fraud Detection System

Capital One Capstone Project

Customer Requirements

As a "customer", I want a way to determine if the incoming transactions on my credit card are fraudulent or not.

- I want the solution to leverage a model which is smart enough to have noticeably better performance than random chance
    - Example: If 50% of transactions were fraudulent, a "dumb" model could mark 50% of transactions correctly simply by flipping a coin and guessing. Your model will need to perform better than this, minimizing both false positives and false negatives.
- I want each transaction to be rated on a scale from 0 to 1 (0 being certainly not fraud, and 1 being most likely to be fraudulent) where the distribution of ratings is uniform (e.g. 20% of transactions score higher than .8 and 30% of transactions score lower than .3 etc.) and be able to configure alerting for my account for a threshold of my choosing
    - Example: User A has a threshold of .6 and wants to be notified of any transactions scored between .6 and 1. User B has a threshold .9 and wants to be notified of any transactions scored between .9 and 1
- I want to be notified whenever a transaction breaches my threshold
    - Note: We have found that the best tool to send fraud notification text messages is **Twilio, not AWS**. They have a large free tier that should cover the needs of this project. Get started here.
- I want to be able to reply to the notification and have the database update the corresponding transaction's fraud flag based on my input
    - For example, the text could say "Reply YES if this was fraud" and when I reply the database will be updated to reflect that
- As part of the project, there will obviously be no integration with actual credit cards. To simulate a purchase, I want a REST API which I can call to add a transaction for a particular user
    - I do NOT want to mark transactions as fraudulent via this API. The API is only for adding transactions to simulate swiping a card at a store
    - The API should have a second endpoint for account creation

Mentor Requirements

AWS / Cloud Services

We want students to gain real world, marketable skills as they progress through this class. All of us mentors have found our AWS experience to be extremely useful and we'd like to pass on to you an opportunity to learn about something that might not be featured in your coursework at UW. It's also cool at the end of the semester when you get to explain to all of your classmates that your project is publicly available on the internet and any of them can try using it in real-time during your demo if they wanted to

AWS, Azure, and GCP are largely interchangeable, but us mentors all work with AWS in our day-to-day which is why we recommend it. We won't be as helpful for debugging purposes if your team chooses to use Azure or GCP instead. AWS is also the most widely used, and therefore the most marketable to future employers.

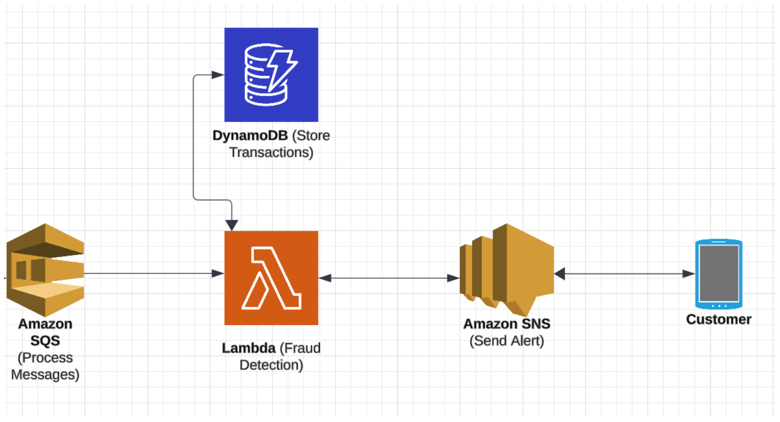
If you choose to use AWS, please reference the documentation in the AWS docs. A few starting references:

- AWS Decision Guides
- What is Lambda?
- Creating Event Driven Architecture with Lambda
- Invoking Lambda with Events

Architecture diagram

We have provided an example architecture diagram as a starting point for your project. These are the general elements we would like to see in your project- a database, some form of an event-driven process, and text-based alerting. Your project will be more intricate than this example and require more elements. **As part of your project's deliverables, we would like to see a detailed architecture diagram** that specifies your project's components and their interactions. Example here.

Note**: Your architecture diagram is a living document**. It should be created when your team first starts design discussions to make sure everyone is on the same page. As your project evolves and changes need to be made, those changes should be reflected in the diagram. Please do not wait until the end of the semester to create a representation of your final product, the diagram is for you, not us. Also- a diagram is a great thing to include in mid-semester and end-of-semester presentations.



Data / Fraud Model

We recommend finding a dataset online which you can use to train/test your model. This will be far easier than creating one yourself. An example dataset can be found here, but there are plenty of other ones on Kaggle if you find something else you think is better. The only requirements are that the dataset has the "normal" information (transaction amount, merchant, location, time, etc) as well as a column that indicates fraud. Additionally, your training dataset and your testing dataset must be distinct. You can't re-test your model on the same transactions you used for training! If you find a dataset you like that only has 1 dataset, you can always use the first 90% for training and reserve the last 10% for testing. Feel free to create any other additional columns for your personal use that you might find helpful, but be sure not to let them impact the model's training as they will have no impact on if the transaction was fraudulent or not.

Helpful Resource: Plagiarism Detection

Here's what we think about a UI

The goal of this project is to gain AWS / cloud skills and become familiar with designing functional project architectures- not to build a UI. We understand that a UI is easy to demo at the end of the semester. But every semester, we see the same version of a UI where students create a page that displays a list of transactions.

If you want to create a UI for this project, we would like it to solely be used as a visual interface to interact with the account/transaction creation API. We do **NOT** want a UI that displays a list of customer transactions where the customer can flag them as fraudulent. We want that functionality to be handled in an event driven manner via email/text.

An example UI we liked in previous semesters: "A totally legit, totally not-a-scam website" where users were asked to give their credit card information… for very legitimate purposes.

Deliverables

We have included a list of items to help you prioritize your work throughout the semester. Make sure that before you move onto “Nice to Have” and “Stretch Goals,” you are confident in your ability to meet the “Must Have” deliverables.

Must Have

- Fraud detection model (details above)
- Live integration with mobile delivery (text)
- Use of cloud computing services
- Architecture Diagram
- Recorded demo of your project in action (mid-semester and end of semester)
- Things we (mentors) could do better next semester :)
- Teamwork
- Fun

Nice to Have

- *Amazing* fraud detection model
- Receiving customer responses and updating the DB based on their response (fraud or not fraud)
- User configurable notification thresholds
- Highly resilient, highly available, scalable, intelligent architecture
    - AWS Cloud Resilience
    - Architecting for Reliable Scalability

Stretch Goal

- *Impeccable* fraud detection model
- UI for creating transactions
- Periodically retraining the model on transactions as you add more
- Performance testing (if not cost prohibitive)