Gil Halevi

Project Report

**Application:**

The github link to my application is as follows:

First, let me explain the technical details of the application. The application is found in the “/app” folder of this submission folder. The application was created using flask. In the app.py file is the function to be run when the user hits each endpoint of the app: “/”,”/customer”,”/customer/see\_remittances”, etc. Every time one hits an endpoint, the function does some computations (potentially querying from the database) and then returns a template, which are the html files representing the pages. The templates are stored in the “/app/templates” folder, and except for the index (home page of the app) are further divided into business and customer (will be explained soon).

The app connects to the database by the Object Relational Mapper (ORM) Flask-Sqlalchemy and the connector mysql-connector. All the models are stored in the models.py file. Queries are issued using the notation Object.query or db\_session.query(Object), with “.filter” corresponding to Select commands and most other commands being named the same way as sql equivalents, although sometimes it can be a bit unclear what SQL query is actually being issued. The database.py file is used to initiate the connection to the database.

Here is the list of use cases from part 3 (as well as a few more), how one would use the application to carry out the use cases, and then what the application is doing. Use case #7 is the one using the machine learning model, so that is where I explain how the integration is done with the machine learning model.

1: A customer wants to see if the insurance gave them a refund. The customer first selects “I am a customer” on the homepage

A screenshot of a red box

Description automatically generated

The customer then enters their SSN on the first page and clicks “Submit”

A screenshot of a computer

Description automatically generated

If the Ssn is not in the system, the page tells them that and they can submit again:

A red and white box with black numbers

Description automatically generated

A screenshot of a computer

Description automatically generated

Upon successful ssn entry, the customer then selects “See Remittances” from the following list.

A screenshot of a computer

Description automatically generated

The customer is then shown a list of all remittances, in chronological order, associated with their account, and can see if the remittance for their claim was given yet.

Under the hood, when the customer enters their Ssn, the app first does a select command to see that the Ssn is valid, and if so saves in session variables the ContractBenefit# of all ContractBenefits (these are benefits associated with specific contracts) for which the customer is the beneficiary. This is done in the customer() function. A screen shot of a computer code

Description automatically generated

When the customer selects See Remittances, the app pulls up all rows in the Remittance table associated with one of those ContractBenefits, ordered by the date they were given

A screen shot of a computer

Description automatically generated

Note also that this is an example of query optimization, since it involves less comparisons to first get the few ContractBenefits that are associated with the user and then select all Remittances associated with one of those ContractBenefit#s, as done here, than it does to do a join first and then pick out all rows with the right ContractBenefit#.

2: A customer wants to see how much they owe. This functions very similarly. As above, the customer clicks “I am a customer” and enters in their Ssn.

A screenshot of a red box

Description automatically generated

A screenshot of a computer

Description automatically generated

They then select “See Invoices” from the following list to see all the invoices associated with their account.

A screenshot of a phone

Description automatically generated

A screenshot of a phone

Description automatically generated

The code is also very similar. The ContractBenefit#s associated with the customer’s ssn are stored in the same way, and the Invocies are queried in the same way, again an example of query optimization:

A computer screen with colorful text

Description automatically generated

3: A customer wants to file a claim. As above, they click “I am a customer” and enter in their Ssn. They then select “File Claim” from the next list.

Next, they are presented with a form to fill out their claims. They enter in all information.

A screenshot of a computer

Description automatically generated

If any of it is invalid, the page tells them and they can enter again

A screenshot of a computer

Description automatically generated

If not, they are booted back to the selection page with text telling them it was successful.

A screenshot of a computer

Description automatically generated

The business can now see this claim in their system (will demonstrate this part in a bit):

A screenshot of a computer

Description automatically generated

This is what the file\_claim function looks under the hood, it queries the two entered parameters to see that they exist in the database and if so it creates a new Claim object and commits it to the system. It automatically creates a new claim\_number (this is the primary key) that doesn’t already exist. It uses the Ssn entered before saved in the session variables for the new created Claim’s Ssn.

A screen shot of a computer program

Description automatically generated

4: A customer wants to make a payment. So they navigate to the selection screen, press make a payment, and then enter in the amount of money and the benefit they are making this payment toward. (this would usually be the deductible). Again, if the information is not right, they will be told.

A screenshot of a computer

Description automatically generated

Otherwise, the money will be added to the Earnings of all associates who have ProductionCredit or a Commission on the given ContractBenefit. Let’s showcase this. First, let’s note that Sitcode 1234567890 has a 15% commission and 10% production credit on ContractBenefit# 1234567890:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

This Sitcode is of the ManagerContract of Associate with AssociateId 1234567890.

A screenshot of a computer

Description automatically generated

Let’s first check the earnings for this associate before the transaction:

A screenshot of a computer

Description automatically generated

Now, after the following payment for 100$:

A close-up of a red box

Description automatically generated

A screenshot of a computer screen

Description automatically generated

This associate’s earnings become raised by 25$, 15$ for the Commission and 10$ for the Commission:

A screenshot of a computer

Description automatically generated

This is done with the following code. As before, the query is optimized by first filtering to the commissions and productioncredits with the right ContractBenefit# before doing a merge with ManagerContract and Associate to find the proper associates. A screen shot of a computer program

Description automatically generated

5: The business wants to process claims. We instead click “I am from the business” on the homepage

A screenshot of a phone

Description automatically generated

And then “See Claims” on the next page

A screenshot of a computer

Description automatically generated

To get to this page. This is how we saw the claim entered before.

A close-up of a document

Description automatically generated

Here is the code for this:

A computer screen shot of text

Description automatically generated

6: An associate passes away and the business needs to bequeath their earnings. We click “Associate Passed Away” on the business select screen and are brought to the following page:

A red box with white text

Description automatically generated

We can enter the associateId and are shown that associate’s earnings and all customers they bequeathed money to, as well as the percent, so that we can send a check to these customers. If the AssociateId is not in the system, we get a message.

A screenshot of a computer

Description automatically generated

Otherwise:

A close-up of a number

Description automatically generated

7: The business wants to recalculate premiums. This is where the ML model is used.

This is done simply by navigating to the business select and clicking “Recalculate Premiums”, and a little prompt shows up to verify that the button was clicked.

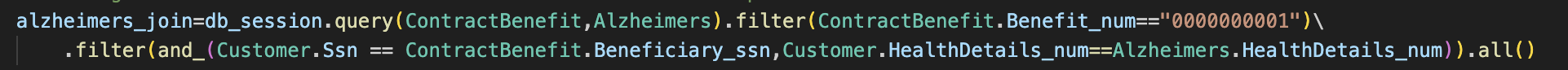
A screenshot of a computer

Description automatically generated

Here is how it works. There are four benefits being tested here, one for diabetes care, one for heart disease care, one for stroke care, and one for Alzheimers care. There are four tables containing medical information for each customer that might be useful to predict that disease: Alzheimers, HeartDisease, Diabetes and Stroke respectively. These contain information like age, sex, cholesterol level, etc.

Additionally, there are machine learning models stored on file in the “/ml\_models” folder. These ml models are trained on data from the data lake to predict the probability that the patient develops the disease in the next year by their health information. Note that the columns in the database are a subset of the columns in the unstructured data lake, as the data is processed. These models are produced by the machine\_learning.ipynb jupyter notebook also in the “/ml\_models” folder (not in “/app”). This notebook can be rerun with new data at any point, allowing for seamless retraining on the newest data.

When one clicks the recalculate premiums button, it calls the recalculate\_premiums() method. This does as follows, one disease at a time, which I will demonstrate with alzheimers. It finds all ContractBenefits that are associated with the Alzheimers benefit. Then it finds the Alzheimers health information, in the Alzheimers table, which is associated with all beneficiaries of that ContractBenefit. Again this query is optimized by first doing the filter to find the ContractBenefits associated with the specific benefit before doing a join with Customer and Alzheimers to get health information for all patients:



The code then loads in the alzheimers model and uses it to predict the predicted risk of developing alzheimers. This risk is used to determine the premium of each ContractBenefit. The premiums in the ContractBenefit table are modified accordingly:

A screen shot of a computer program

Description automatically generated

The same is done with each of the other diseases.

Let’s showcase this. We can view the premiums by clicking “Show Premiums” on the business select screen. I have manually set a few of these premiums to 0

A close-up of numbers

Description automatically generated

A close-up of a number

Description automatically generated

A number of digits on a white background

Description automatically generated

A number of digits on a white background

Description automatically generated

Then, after clicking recalculate premiums, these premiums get set:

A number of numbers on a white background

Description automatically generated

A number of black numbers

Description automatically generated with medium confidence

A number of digits on a white background

Description automatically generated

A number of digits on a white background

Description automatically generated

A number of digits on a white background

Description automatically generated

Use case 7 would be rerun on an annual basis as premiums are updated, and would also be run for new customers as they enter the system.

Obviously, these are not the only use cases that would be associated with an insurance business, but they hopefully give a good overview of how the system works and it can be easily expanded to include more use-cases. I did not know flask or sqlalchemy before this project, so it took a lot of time and effort to code up this app and I hope my effort shows :)

**Reference Architecture:**

The vision for this reference architecture is to allow the company to most efficiently and effectively carry out the functions of an insurance business. Some foundational principles are as follows:

* IT solutions should be designed to best serve the customers and the business
* IT solutions should support and enable data-driven decision-making, providing accurate and timely insights to the business
* IT solutions should not be overly complex, making the business more agile, flexible and accessible
* Users must abide by governance as they use IT solutions

The organizing framework is that there is a business domain, which provides the use-cases for the insurance business laid out in part 3, an information domain that organizes and stores the data (this is the data lake and the database), and an application domain (this is the flask app discussed above) in the middle that allows the business to carry out their use cases using the data.

The methods, guidelines and policies to plan, deliver and operate business solutions are as follows. On the application side, each page and UI component should be designed with one or more specific use cases in mind. For each use case, it should be clear which UI components one would use to complete it. The UI should follow a consistent theme so as to maintain the professionalism of the business. The UI should be separated from the application logic. The application logic should be clear and well-documented to meet the requirements and minimize errors. On the information side, the database should store all the data produced by the company, as well as all data needed for day-to-day operation. Data that is commonly accessed together should be stored together. As much as possible, data should not be duplicated in multiple places. BCNF should be achieved where possible.

Some aspects of data governance are as follows. Data from the data lake must not be used to train the machine learning model until it is assured that the data is processed – continuous variables need to be normalized, all variables for which there is missing information must be set to the average value, and sparse columns are removed to avoid the effect of noise. In real life, the machine learning model itself must not use medical details on which it is against the law to discriminate on, although I may have violated this here (Prof. Franchitti said it is ok for this project). Additionally, the insurance company must not store medical information that is private and has not been shared with it. When a customer leaves the system, their information, including all their medical information, should be deleted from the system.

The diagram for the reference architecture can be seen here:

