

Final Project Report – Part 3 & 4

Name: Sumanth Reddy Muni

Date: 05/17/2021

Section: MSIS - Summer 2021

Database Systems Final Project Report Part 3 & 4

Assignment Layout (25%)

- o Assignment is neatly assembled on 8 1/2 by 11 paper.
- o Cover page with your name (last name first followed by a comma then first name), username and section number with a signed statement of independent effort is included.
- o Answers to textbook exercises in Question 1 are correct.
- o File name is correct.

Answers to Individual Questions:

(100 points total, all questions weighted equally)

- o Assumptions provided when required.

Total in points (100 points total): _____

Professor's Comments:

Affirmation of my Independent Effort: Sumanth Reddy Muni
(Sign here)

I. Introduction:

In the current state, the business operates with multiple data stores with each department of the business using their own different solutions (database and software applications) to store and manage their data. Having so many data stores and separate processes to manage has made it very difficult for our fictional enterprise to manage their data as there has been multiple instances where data was duplicated or data was not tracked in some systems. This inconsistency has made the business to rethink their data storage strategy. In this project, we will work on centralizing the data storage process by redesigning the schema from bottom-up by collecting all the existing relational schemas from all the departments and merging them into a single schema so that we can make sure such inconsistencies don't occur in the database in the future.

- In part 1, I created a conceptual model by understanding the use-cases and enterprise blueprint.
- In part 2 of the project, I focus on Creating and optimizing a logical database schema for the structured data used by the insurance company at hand based on the conceptual model created earlier. I also put together a small data lake to compile various relevant datasets and identify how insights that may be extracted from the data collected will feed into the EDA that was designed in the first part of the project. We also use AWS cloud platform to build a data lake whose design has been explained in detail later in this report.
- Here, In part 3, we put all this into practice. First, I refined the database relations further to choose a part of the enterprise that we would first concentrate on. I chose to implement Quote request workflow and the database contains all the relevant tables to support this workflow. I have implemented 2 different apps as part of this project:
 - The *first app* is a customer facing app where the customer can request for a quote by providing some information.
 - The *second app* is a customer support dashboard app internal to the organization which the Insurance company's customer support associates can use it to check the quote requests and use the information to call the customer to convert the quote request to a sale. Frontend was built using HTML, CSS, JavaScript, jQuery and Bootstrap and Backend was built using Java, Spring MVC, Spring Data,

JDBC, **Hibernate ORM**, Python and Flask. MySQL Database was used to store the data. Normalization, Indexing & Stored Procedures were used to improve the performance of the queries. Also, an **ORM framework** was used to help with mapping the relationship between java objects in the frontend. Finally, I also built a Machine learning model which predicts if a user might get Chronic Disease. This Machine learning model was deployed and exposed as Python Flask API. Based on the data filled by the user, the frontend web app makes a request to this prediction API and tells the user if they are eligible for a discount.

Link to the Github repository:

https://github.com/sumanthreddym/dms_final_project

II. Logical Schema Modeling

A. Modeling procedure

In first part of this project, we started by creating the conceptual model, where we analyzed the blueprint given in the slides to create the conceptual model. In this part of the project, we further optimize the conceptual model, normalize, refine relationships to create a logical model that also includes a few more relations so that the insurance company can collect data to help forecast chronic diseases.

Similar to the Conceptual model created earlier, each department's entities are given a color code for easy readability. I analyzed the business use case further to add data types for each of the attributes. Then Primary Key, Foreign Key relationships were added to make the relational schema ready to be used for the physical model.

LucidChart(<https://www.lucidchart.com/pages/>) was the tool used for creating the Logical diagram.

B. Important points from the modeling process

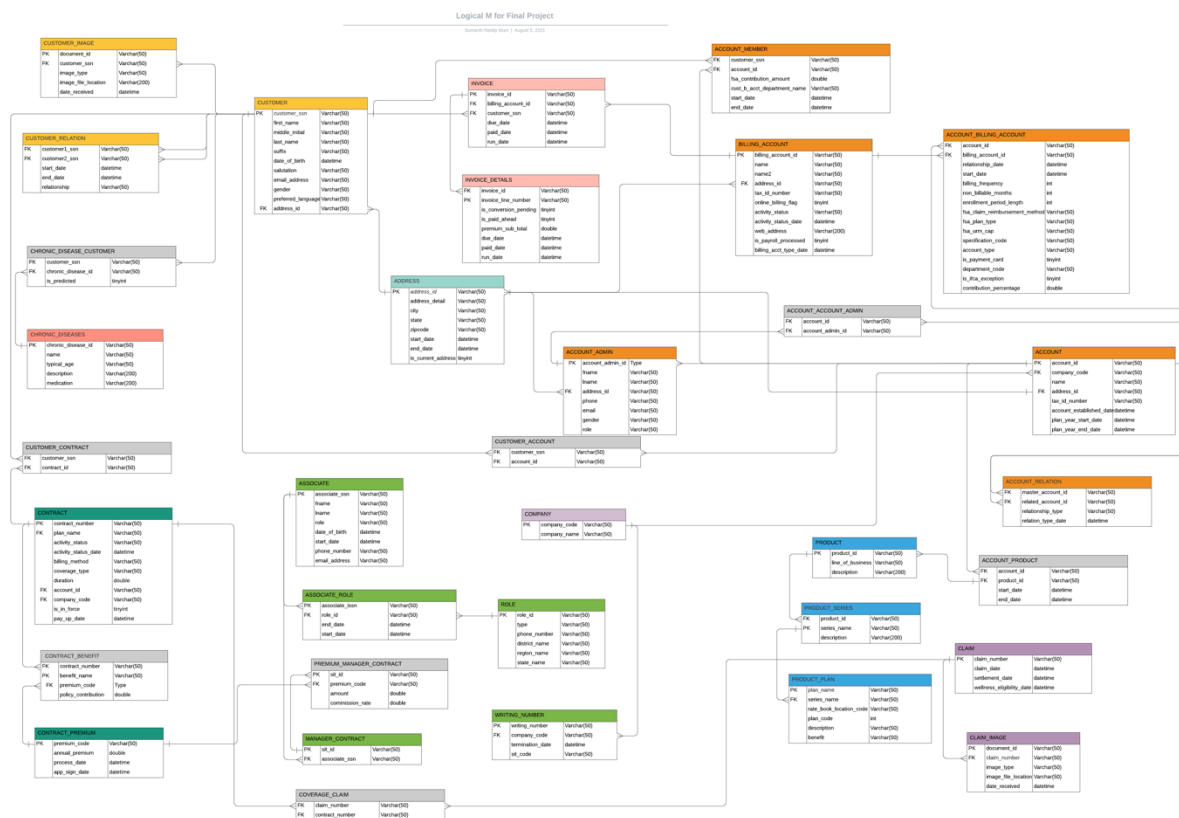
- Many to many relationships are converted to 2 one to Many relationships in many cases. I tried to get all database relations to BCNF.
- In some cases, we use a combination of foreign keys An Address relation was created to extract away the attributes that were common among many different tables and a unique id address_id was created to uniquely identify each different address. This will lead to better manage the address as well as store multiple addresses for the same customer. For instance, if a customer moves from one home to another, we can have both the addresses saved for our future references.
- A few other tables were normalized to get them to BCNF. So, all tables now satisfy 1NF, 2NF, 3NF and BCNF.
- To store details about customer, we have CUSTOMER, CUSTOMER_IMAGE(each customer can have multiple images) and CUSTOMER_RELATION entities.
- Each customer can have many accounts and each account can have many customers associated with it.
- To store details about Account, we have ACCOUNT, ACCOUNT_MEMBER, ACCOUNT_ADMIN, ACCOUNT_BILLING_ACCOUNT and BILLING_ACCOUNT entities.
- Account can have multiple billing accounts and a billing account can have multiple accounts related with it. This has been modeled by creating a new entity ACCOUNT_BILLING_ACCOUNT.
- To store details about Invoice, we have INVOICE, INVOICE_DETAILS entities.
- Customer can have many Invoices. An invoice is related to only one customer. An Invoice is billed to a particular billing account.
- To store details about Product, we have PRODUCT, PRODUCT_SERIES and PRODUCT_PLAN entities.
- Each product is associated to multiple accounts and each account is associated with multiple products. This has been modeled by creating a new entity
- To store details about a contract, we have CONTRACT, CONTRACT_BENEFIT, CONTRACT_PREMIUM entities.
- To store details about Associates, we have ASSOCIATE, ASSOCIATE_ROLE, ROLE, MANAGER_CONTRACT and WRITING NUMBER entities. Associate types and the region of coverage that they are responsible for can be tracked using the ASSOCIATE_ROLE entity. Each MANAGER_CONTRACT has

multiple WRITING_NUMBERS. Some of these entities has been dropped in this part of the project since it doesn't have much significance here.

- Each ACCOUNT has multiple associates, admins and members.
- To process CLAIMs, we have CLAIM and CLAIM_IMAGE entity that are related to CONTRACT entity through the COVERAGE_CLAIM entity. This entity has been dropped in this part of the project since it doesn't have much significance here.

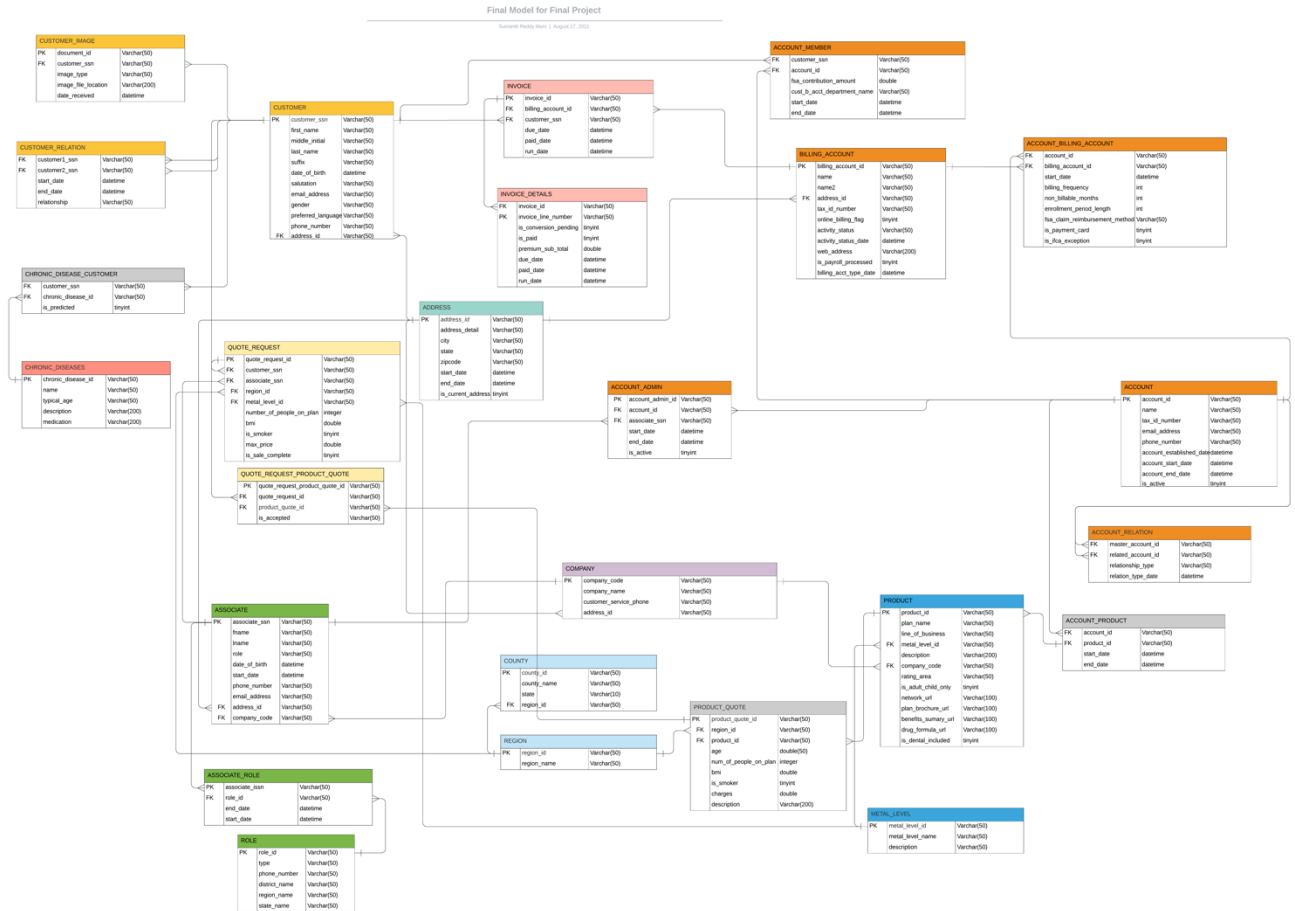
C. Result of Modeling process

Logical model created in part 2 of the project(previous version):



Updated ER diagram(new version):

Following ER diagram was modeled to include only the entities relevant to quote request application(For better readability of the model, a PDF has also been



The above ER diagram is optimized and includes insights extracted from data collected in the data collection phase. Here we have added CHRONIC_DISEASES relation along with CHRONIC_DISEASE_CUSTOMER relation. CHRONIC_DISEASES relation has data about all possible chronic diseases that we extract from unstructured data. CHRONIC_DISEASE_CUSTOMER relation includes data which says whether a customer has a chronic disease or if our analysis from our data lake predicted that a customer might have a chronic disease in future.

Some important business terms:

- Each Product(aka plan) is available in multiple Regions. This is what the users purchase from our Health Insurance company.
- Each Region is made up of multiple counties.

- Companies refer to all our subsidiaries or partners who sell their insurance through us.
- An Associate works for one or more of our companies/partners.
- An Associate can work in multiple roles.
- An Associate is assigned to a particular Quote Request so that he can help the customer make a better decision and possibly sell the insurance product.
- Insurance product has multiple metal levels such as Gold, Silver, Bronze, Platinum.
- Each customer can have multiple accounts with us, and each account is assigned to an Associate.
- An account can have multiple billing accounts which also means that an account can have multiple customers.
- Invoice is the bill generated that is associated with an Account.

Following were created on MySQL to optimize the queries:

- Indexing
- Stored Procedures
- Normalization
- Transactions were used in the Java application in the service layer to ensure that we are ACID compliant.

We can import this schema to MySQL using the SQL script file "database_schema.sql".

III. Compilation of Unstructured and Structured Datasets

Following datasets were collected to be added to our data lake:

- U.S. Chronic Disease Indicators, Dataset by CDC at <https://chronicdata.cdc.gov/Chronic-Disease-Indicators/U-S-Chronic-Disease-Indicators-CDI-/g4ie-h725>
- Kidney Chronic Disease indicators, Dataset by UCI - https://archive.ics.uci.edu/ml/datasets/Chronic_Kidney_Disease

- 2020: Complete NHANES US Health Data – Unstructured Dataset by CDC at <https://wwwn.cdc.gov/nchs/nhanes/>
- Asthma Emergency Department Visit Rates, Dataset by [CALIFORNIA HEALTH AND HUMAN SERVICES](#) at <https://data.world/chhs/34f3464e-b2eb-4f74-9ef9-f378711aa0f5>
- Chronic illness: symptoms, treatments and triggers, Dataset by Flaredown at <https://www.kaggle.com/flaredown/flaredown-autoimmune-symptom-tracker>
- Stroke Prediction Dataset, Dataset by [fedesoriano](#) at <https://www.kaggle.com/fedesoriano>
- Heart Failure Prediction, Dataset by [Larxel](#) at <https://www.kaggle.com/andrewmvd/heart-failure-clinical-data>
- Diabetes Data Set, Dataset by [Vikas Ukani](#) at <https://www.kaggle.com/vikasukani/diabetes-data-set/>
- Parkinsons disease Dataset, Dataset by [Vikas Ukani](#) at <https://www.kaggle.com/vikasukani/parkinsons-disease-data-set>
- Chronic Kidney Disease, Dataset by [Co-learning Lounge](#) at <https://www.kaggle.com/colearninglounge/chronic-kidney-disease>

All the datasets listed above are a combination of unstructured and structured data that can be used by our fictional insurance company to help forecast chronic diseases based on various factors.

IV. Creating Full Stack app, building a Data Lake on AWS and including insights extracted from data collected using Machine Learning

A. Building a Full Stack app:

Technologies Used:

- **Frontend:**
 1. HTML
 2. CSS

3. JavaScript
4. Bootstrap
5. jQuery

- **Backend:**

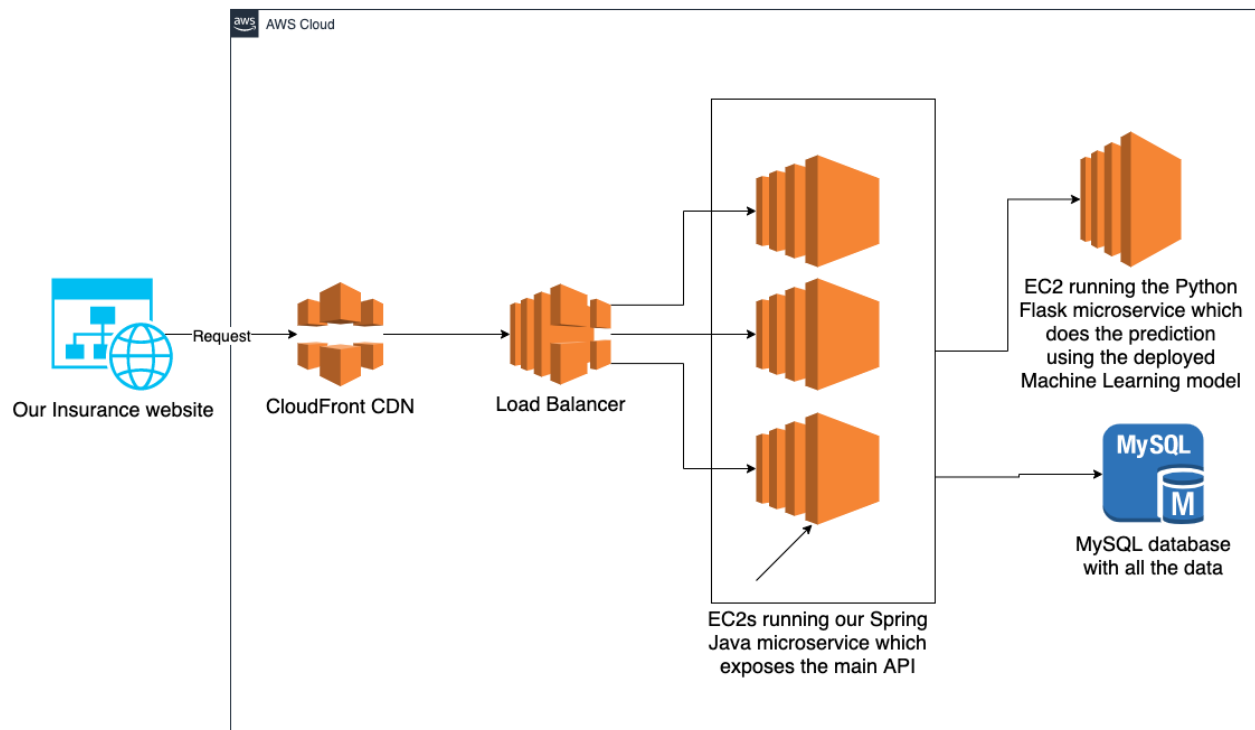
1. Java
2. Spring MVC
3. Spring JPA
4. Spring Data
5. *Hibernate ORM (ORM Framework)*
6. JDBC
7. Python
8. Flask

- **Database:**

1. MySQL

Architecture:

Architecture



Instructions(How to Run the apps?)

To run the Java app, open the project in IntelliJ and run the main method. Since, we are using Spring Boot, it contains Tomcat internally built to run the app. Maven is used for dependency management. Use “maven clean install” command to download dependencies. Now, open web browser and type “localhost:8080” to visit the index page of the user app. For customer-facing app, use the address “localhost:8080/support_dashboard”.

Create the database in MySQL using the SQL script “database_schema.sql”.

Set the following Spring properties in application.properties in the Java project. These properties are used by the Java app to connect to the database.

```
spring.datasource.url=jdbc:mysql://localhost/INSURANCE_COMPANY?useSSL=false&serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=YourPasswordHere
```

To run the Python app, open the project and run the app.py python file. This will run the Flask app and expose the “localhost:5000/predict_kidney_disease” API.

Some available APIs:

GET /customers – List customer

POST /customers – Create customer

POST /prediction – Predict if customer is vulnerable to chronic disease

GET /quotes – List Quotes

GET /quote_requests – List Quote requests

POST /quote_requests – Create Quote request

GET /quote_request_product_quotes – List Recommended Quotes

Following is the structure of the project:

```
src
  main
    java
      com.sumanth.healthinsurance
        dto
        model
        repository
```

```

        service
        web
            controllers
            Main class(Spring boot app initialization)
python
    app.py(Flask app with ML model)

resources
    static
        css
        images
    templates
        index.html(thymeleaf)
    application.properties
```

database_schema.sql(MySQL Database creation script)
chronic_disease_notebook.ipynb

Hibernate ORM:

Hibernate ORM was used to map the relationships between tables on the Java side.

Following are some of the features that were implemented:

Customer facing web app:

- User can fill in personal details in the frontend app to request for a quote. The app returns a list of quotes personalized to user's preferences. It also lists the customer support associate's number that a user can use to call us for help. Following images list each step of the process as seen by the customer in the UI.

1. Customer fills General information using the following page and clicks next to goto step 2:

GET HEALTH INSURANCE QUOTE

If you want to explore the best health insurance plans offered by some amazing companies, you are at the right place!
Fill all the fields below to go to the next step

General Personal Finish

General Information: Step 1 - 3

Which state do you live in?: *

PA

What is your zipcode?: *

16803

Gender: *

Male

Date of birth(MM/DD/YYYY): *

02/01/1989

Do you use Tobacco?: *

Yes

How many dependents?: *

1

Which Metal Level do you prefer?: *

Gold

Next

Click here to check if you are eligible for offer!

2. Customer fills Personal information using the following page and clicks next to goto step 3:
- 3.

sumanthreddy@mac: ~\$ x Sumanth Health Insurance x Untitled - Jupyter Notebook x +

localhost:3000

NTU, AI, Data Structures, Codepath, DB, Jenkins, Adobe Commerce, Http://www.vista... 2022

Other Bookmarks Reading List

Sumanth Health Insurance Home About Us Contact Us

GET HEALTH INSURANCE QUOTE

If you want to explore the best health insurance plans offered by some amazing companies, you are at the right place!

Fill all the fields below to go to the next step

General Personal Finish

Personal Information: Step 2 - 3

First Name: *

Nick

Last Name: *

Van Howe

Contact No.: *

8765678998

SSN: *

345686788

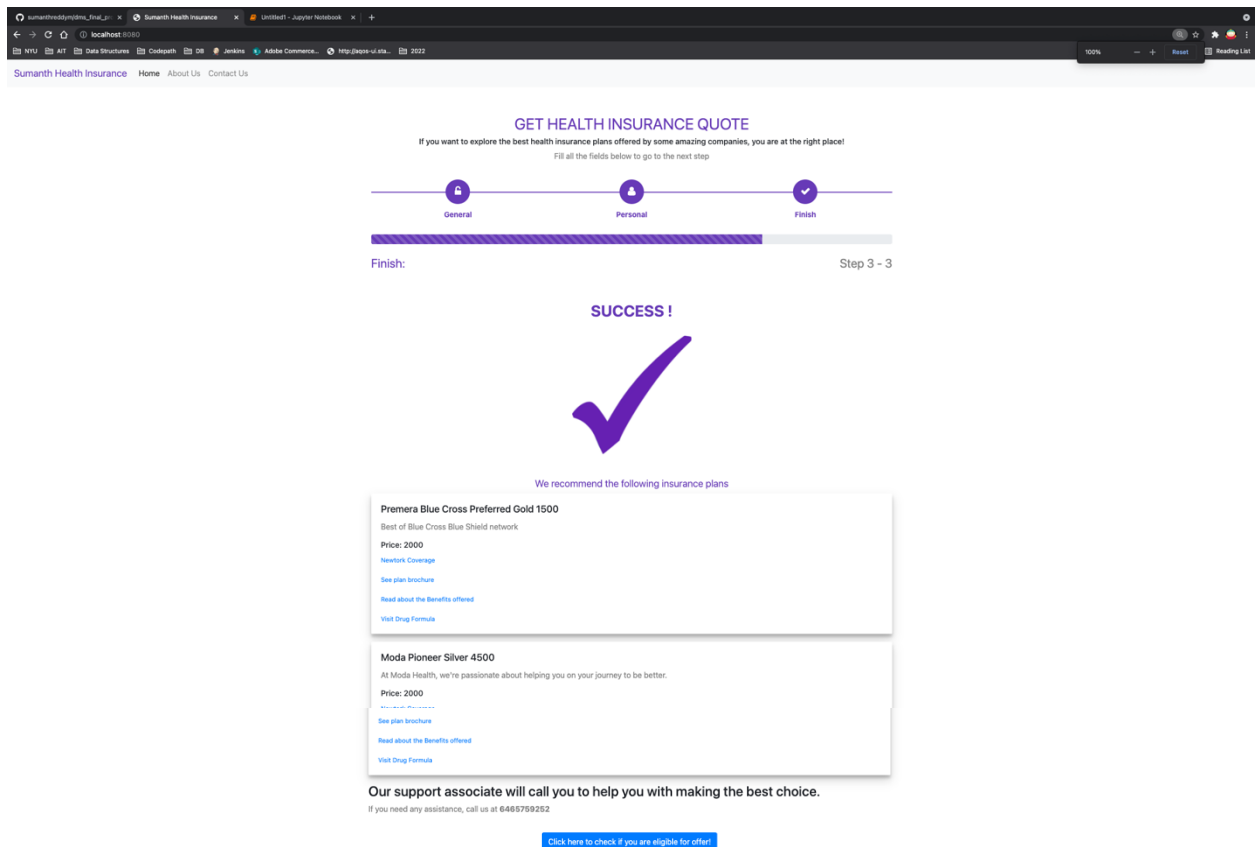
Email Address: *

nick@google.com

Previous Next

Click here to check if you are eligible for offer!

- Customer is recommended a list of insurance plans that are tailored to the information that the customer entered in the previous 2 steps.



- Customer can click on the “*Click here to check if you are eligible for offer button*” to check if they are eligible to get an offer. On clicking this button, customer is asked to fill another form which collects data that can be used by our **Machine Learning** model to predict if the customer might be vulnerable to chronic disease in the future. If the customer is predicted to be vulnerable, then the app will tell the customer that they are not eligible for a discount. Else, the app gives the customer a discount that is inversely proportional to the risk. Following are the 2 cases as seen by the customer.
1. Customer fills the form and clicks on the “Check Eligibility” button.

Want a discount?



Please fill out the following form to see if you are eligible for the offer.

Note: The data that you provide will be used by our machine learning model to check if yo qualify for an offer. By clicking on "Check Eligibility" button below, you agree to our data usage policy.

Age: *	<input type="text" value="Age"/>
Blood Pressure: *	<input type="text" value="bloodPressure"/>
Specific Gravity: *	<input type="text" value="1.005"/>
Albumin: *	<input type="text" value="0"/>
Sugar: *	<input type="text" value="0"/>
Red Blood Cells: *	<input type="text" value="normal"/>
Pus Cell: *	<input type="text" value="normal"/>
Pus Cell Clumps: *	<input type="text" value="normal"/>
Bacteria: *	<input type="text" value="present"/>
Blood Glucose Random: *	<input type="text" value="bloodGlucoseRandom"/>
Blood Urea: *	<input type="text" value="bloodUrea"/>
Serum Creatinine: *	<input type="text" value="serumCreatinine"/>
Sodium: *	<input type="text" value="sodium"/>
Potassium: *	<input type="text" value="potassium"/>
Haemoglobin: *	<input type="text" value="haemoglobin"/>
Packed Cell Volume: *	<input type="text" value="packedCellVolume"/>
White Blood Cell Count: *	<input type="text" value="whiteBloodCellCount"/>
Red Blood Cell Count: *	<input type="text" value="redBloodCellCount"/>
Hypertension: *	<input type="text" value="Yes"/>
Diabetes Mellitus: *	<input type="text" value="Yes"/>
Coronary Artery Disease: *	<input type="text" value="Yes"/>
Appetite: *	<input type="text" value="Good"/>
pedal Edema: *	<input type="text" value="Yes"/>
Anemia: *	<input type="text" value="Yes"/>

Close

Check Eligibility

2. **Case 1:** No discount since the Machine Learning model predicted that the customer might get Chronic Disease.

The screenshot shows a web application interface with a central modal titled "Want a discount?". The modal contains a form for medical data entry. The background is a dark grey with a purple header and footer. The header has a "General" tab and a "Finish" button. The footer has a "Preferred Gold 1500" section and a "4500" section. The modal is a white box with a close button in the top right corner. It contains a disclaimer about data usage, a list of medical inputs, and a "Check Eligibility" button. The inputs are as follows:

Field	Value
Age	70
Blood Pressure	130
Specific Gravity	1.005
Albumin	1
Sugar	3
Red Blood Cells	normal
Pus Cell	normal
Pus Cell Clumps	abnormal
Bacteria	present
Blood Glucose Random	2
Blood Urea	2
Serum Creatinine	23
Sodium	23
Potassium	12
Haemoglobin	12
Packed Cell Volume	43
White Blood Cell Count	100
Red Blood Cell Count	100
Hypertension	Yes
Diabetes Mellitus	Yes
Coronary Artery Disease	Yes
Appetite	Poor
pedal Edema	No
Anemia	Yes

Below the inputs, the modal displays the message: "Sorry, our Machine Learning model thinks you are not eligible for the offer." At the bottom of the modal are two buttons: "Close" and "Check Eligibility".

3. **Case 2:** Discount offered to customer since the **Machine Learning model** predicted that the customer might NOT get a Chronic Disease in future.

Want a discount?



Please fill out the following form to see if you are eligible for the offer.

Note: The data that you provide will be used by our machine learning model to check if you qualify for an offer. By clicking on "Check Eligibility" button below, you agree to our data usage policy.

Age: *	<input type="text" value="25"/>
Blood Pressure: *	<input type="text" value="120"/>
Specific Gravity: *	<input type="text" value="1.020"/>
Albumin: *	<input type="text" value="0"/>
Sugar: *	<input type="text" value="1"/>
Red Blood Cells: *	<input type="text" value="abnormal"/>
Pus Cell: *	<input type="text" value="abnormal"/>
Pus Cell Clumps: *	<input type="text" value="normal"/>
Bacteria: *	<input type="text" value="present"/>
Blood Glucose Random: *	<input type="text" value="121"/>
Blood Urea: *	<input type="text" value="40"/>
Serum Creatinine: *	<input type="text" value="1.2"/>
Sodium: *	<input type="text" value="120"/>
Potassium: *	<input type="text" value="4.9"/>
Haemoglobin: *	<input type="text" value="15.4"/>
Packed Cell Volume: *	<input type="text" value="36"/>
White Blood Cell Count: *	<input type="text" value="3000"/>
Red Blood Cell Count: *	<input type="text" value="5"/>
Hypertension: *	<input type="text" value="No"/>
Diabetes Mellitus: *	<input type="text" value="No"/>
Coronary Artery Disease: *	<input type="text" value="No"/>
Appetite: *	<input type="text" value="Poor"/>
pedal Edema: *	<input type="text" value="No"/>
Anemia: *	<input type="text" value="Yes"/>

Thanks for your patience.
Congratulations! Our machine learning model says that you are eligible to receive discount. You can receive a discount of 20% on the original cost of insurance.

Close

Check Eligibility

If you want to explore the b

, you are at the right place!



General

General Information:

Which state do you live in?: *

NY

What is your zipcode?: *

Zip Code

Gender: *

Male

Date of birth(MM/DD/YYYY): *

mm/dd/yyyy

Do you use Tobacco?: *

Yes

How many dependents?: *

Number of dependents

Which Metal Level do you prefer?: *

Gold



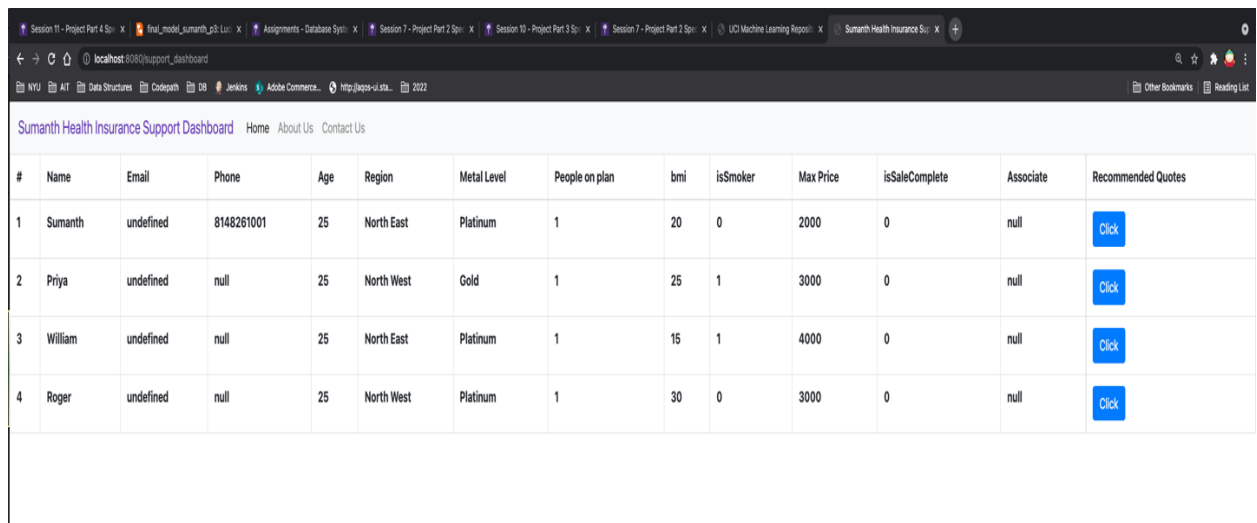
Finish

Step 1 - 3

Next

Customer Support app:

- Customer Support associate logs in to the customer support dashboard where they see a list of pending quote requests.



#	Name	Email	Phone	Age	Region	Metal Level	People on plan	bmi	isSmoker	Max Price	isSaleComplete	Associate	Recommended Quotes
1	Sumanth	undefined	8148261001	25	North East	Platinum	1	20	0	2000	0	null	Click
2	Priya	undefined	null	25	North West	Gold	1	25	1	3000	0	null	Click
3	William	undefined	null	25	North East	Platinum	1	15	1	4000	0	null	Click
4	Roger	undefined	null	25	North West	Platinum	1	30	0	3000	0	null	Click

- Customer support associate clicks on the button corresponding to the quote request that they want to act on. The app displays a list of possible quotes that the customer may be interested in. When the associate calls the customer, this information will help the associate make a sale or assist the customer with their questions. It also lists links to policy documents for each of the plans which can be used by the associate to refer if they need more information to assist the customer.

	Metal Level	Max Premium
	Platinum	2000
	Gold	3000
	Platinum	4000
	Platinum	3000

Recommended Quotes

Please call the customer and recommend the following quotes.

Premera Blue Cross Preferred Gold 1500

Charges: 2000

Is Dental Included? 1

Is Adult only? 0

[Newtork Coverage](#)

[See plan brochure](#)

[Benefits offered](#)

[Drug Formula](#)

Moda Pioneer Silver 4500

Charges: 2000

Is Dental Included? null

Is Adult only? 1

[Newtork Coverage](#)

[See plan brochure](#)

[Benefits offered](#)

[Drug Formula](#)

Close

Mark Resolved

B. Predicting if a customer might get a Chronic disease using Machine Learning:

We use Machine Learning to predict if a customer might get Chronic Disease in the future. The dataset available at https://archive.ics.uci.edu/ml/datasets/Chronic_Kidney_Disease was used for this task.

Jupyter notebook contains the code for this ML task. We start by cleaning the data where nulls are filled with random numbers and duplicate rows are dropped and columns are renamed. We also do a feature selection where we drop the features that are not very well correlated. A few visualizations are shown as part of data exploration and then we use Decision Tree Classifier model to predict if the customer is vulnerable to Chronic Kidney Disease in the future. This model was further deployed and exposed as a Python Flask API. This in itself is a microservice that returns 1 if the user is vulnerable to chronic disease and 0 otherwise. The Spring Java application which is the user facing API makes an API call to this Python Flask API to get the result of prediction. Following image shows the result generated by our model.

```
In [180]: X = df[[col for col in df.columns if col != 'class']]
          y = df['class']

In [181]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30, random_state = 0)

In [182]: decision_tree_classifier = DecisionTreeClassifier()
          model = decision_tree_classifier.fit(X_train, y_train)

In [183]: print("Accuracy:" + str(accuracy_score(y_test, decision_tree_classifier.predict(X_test))))

Accuracy:0.9583333333333334
```

Following is the python flask API code snapshot:

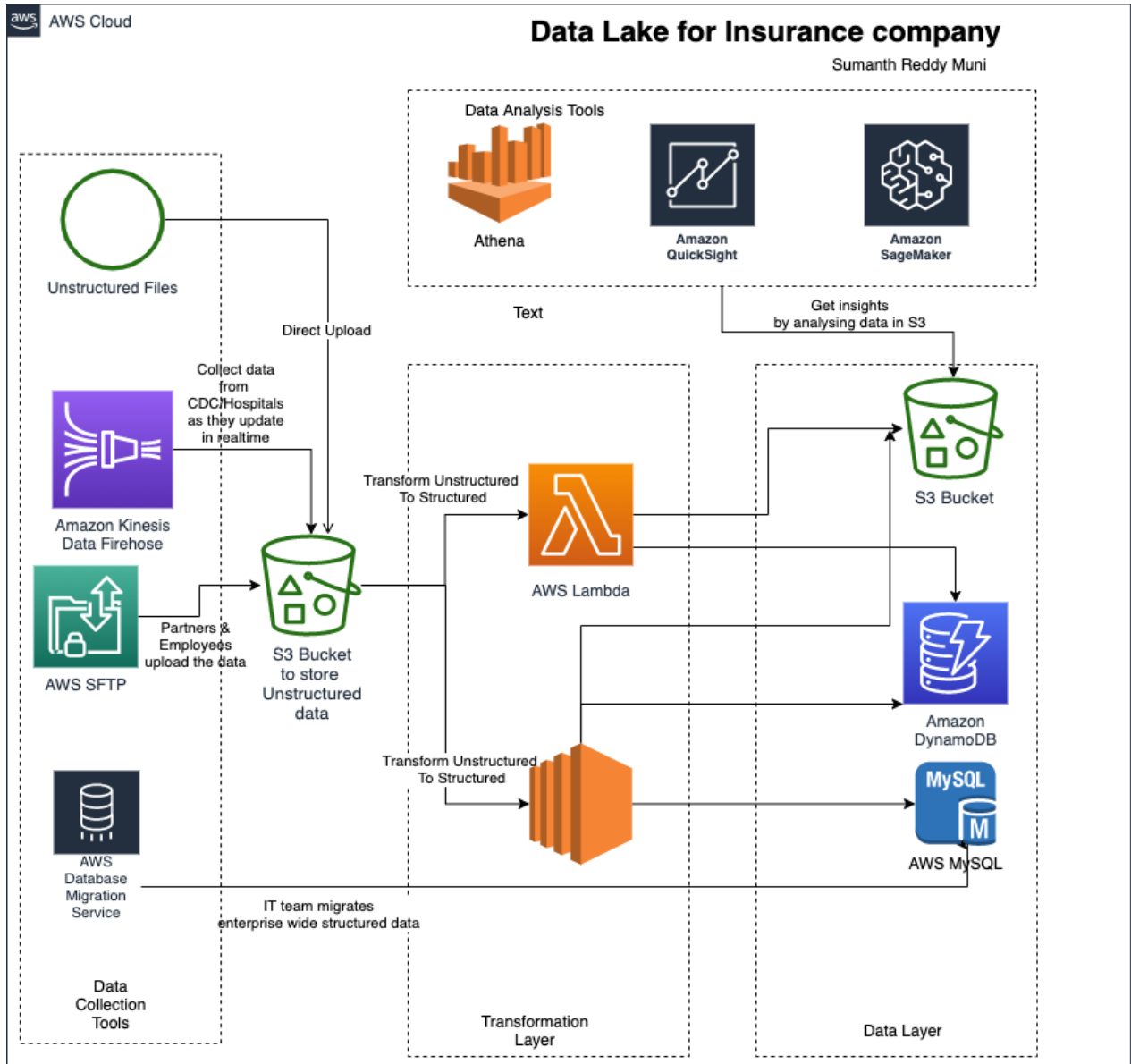
```

1  import os
2  import joblib
3  from flask import Flask, jsonify, request
4  from flask_restful import Api, Resource
5
6  app = Flask(__name__)
7  api = Api(app)
8
9
10 class PredictKidneyDisease(Resource):
11     @staticmethod
12     def post():
13         data = request.get_json()
14
15         model = joblib.load('chronic_disease_kidney.model')
16         prediction = model.predict([[data['age'], data['bloodPressure']],
17                                     return jsonify(str(prediction[0]))
18
19
20 api.add_resource(PredictKidneyDisease, '/predict_kidney_disease')
21
22 if __name__ == '__main__':
23     app.run(debug=True)

```

C. Datalake:

Following data lake design was created after analyzing all the datasets collected and also taking into account the scope of the project as well as the requirements of the insurance company:



Here, we use AWS Cloud to create our data lake. The data that we collected can directly be uploaded to AWS S3 bucket (let us name it unstructured bucket) using a web browser or AWS CLI commands. Our Partner Hospitals, Clinics, Insurance companies can upload unstructured data such as new patient records or survey forms securely to the AWS S3 bucket using AWS SFTP. We can use AWS Database migration service to migrate the structured data that was stored by all the departments in different clouds/data centers to AWS MySQL so that we have all data that belongs to all departments of our organization are stored in one database in AWS. If CDC/Partner hospitals update their database/website, then

we can stream such data using webhooks to Kinesis Firehose. Data from Firehose can be uploaded to S3 bucket using an AWS Lambda function.

Now, our AWS S3 bucket that contains the unstructured data can be transformed automatically as an object is placed into the bucket using a event trigger that would automatically call an API in our web API hosted in EC2 or automatically trigger a Lambda function.

We can store our structured data required for Online Transaction Processing in MySQL and Amazon DynamoDB. Some data that we don't need for Online Transaction Processing can be stored in S3 bucket for further analysis by Big Data/Machine Learning tools in AWS such as Amazon Athena, Amazon QuickSight and Amazon Sagemaker. Further, to perform non transactional queries that are required by managers, AWS Athena was used.

Link to the Github repository:

https://github.com/sumanthreddym/dms_final_project

V. Conclusion

We built a Full Stack web app where the user can get health insurance quote requests by providing some information. We also built a dashboard app which can be used by the Customer Support associates to better assist our customers in buying insurance policies. We updated the logical model to create the final ER model that has the most important entities and attributes required to build our Insurance Quote Request website. MySQL was used to store the information. Database queries were optimized by following some best practices. A machine learning model was trained and deployed which helped us predict if the customer might get a chronic disease in the future. If the customer could get chronic disease in future, we didn't offer them a discount. This should help us reach more healthy customers which would in turn reduce risks and increase profits for our insurance company.

VII. References:

Following material was used for the project and this report write up:

- Professor's slides

- Lucid chart web app
- Course textbook
- Project material
- U.S. Chronic Disease Indicators, Dataset by CDC at <https://chronicdata.cdc.gov/Chronic-Disease-Indicators/U-S-Chronic-Disease-Indicators-CDI-/g4ie-h725>
- 2020: Complete NHANES US Health Data – Unstructured Dataset by CDC at <https://wwwn.cdc.gov/nchs/nhanes/>
- Asthma Emergency Department Visit Rates, Dataset by [CALIFORNIA HEALTH AND HUMAN SERVICES](#) at <https://data.world/chhs/34f3464e-b2eb-4f74-9ef9-f378711aa0f5>
- Chronic illness: symptoms, treatments and triggers, Dataset by Flaredown at <https://www.kaggle.com/flaredown/flaredown-autoimmune-symptom-tracker>
- Stroke Prediction Dataset, Dataset by [fedesoriano](#) at <https://www.kaggle.com/fedesoriano>
- Heart Failure Prediction, Dataset by [Larxel](#) at <https://www.kaggle.com/andrewmvd/heart-failure-clinical-data>
- Diabetes Data Set, Dataset by [Vikas Ukani](#) at <https://www.kaggle.com/vikasukani/diabetes-data-set/>
- Parkinsons disease Dataset, Dataset by [Vikas Ukani](#) at <https://www.kaggle.com/vikasukani/parkinsons-disease-data-set>
- Chronic Kidney Disease, Dataset by [Co-learning Lounge](#) at <https://www.kaggle.com/colearninglounge/chronic-kidney-disease>
- Hibernate ORM - <https://hibernate.org/orm/>
- Spring MVC - <https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html>
- Spring Data JPA - <https://spring.io/projects/spring-data-jpa>
- Joblib - <https://joblib.readthedocs.io/en/latest/>
- Scikit learn - <https://scikit-learn.org/>
- Pandas - <https://pandas.pydata.org/>

Link to the Github repository:

https://github.com/sumanthreddym/dms_final_project