Final Documentation

Team DivideByZero

CS410 Text Information Systems

NetID	Name	Email	Role
connorb8	Connor Brown	connorb8@illinois.edu	
frankns2	Frank Salamone	frankns2@illinois.edu	Captain
jt38	Jonathan Trusheim	jt38@illinois.edu	
raylb2	Ray Butler	raylb2@illinois.edu	
xhuang84	Xinlei Huang	xhuang84@illinois.edu	

Medical professionals must select, interpret, and act on the information contained in a large number of documents for each clinical interaction. When seeing a patient for the first time, a clinician may need access to only a small subset of the documents in the patient's EMR chart. The makeup of this subset of documents would depend on the type of clinician and type of visit. An endocrinologist seeing a hypothyroid patient for the first time may want to see all CT scans, lab tests, and clinical notes related to "thyroid disease". Finding relevant documents in a chart is particularly difficult because there are many documents related to billing, compliance, and other non-clinically relevant information contained in the chart.

In the documentation below, we will detail how team DivideByZero has implemented a search engine to search through simulated medical charts.

1. Overview of function of the code

At a high level, our code can be divided into 3 parts:

a. Chart creation.

As the use of real medical data is problematic, we created a dataset consisting of several 10,000 artificial charts. Our system is designed to be scalable so that it may be applied to a larger dataset in a real-world application.

The data and scripts for creating our artificial charts are contained in the folder chartmaker.

b. Back end – search.

We use elasticsearch to implement our search engine. Elasticsearch instances are deployed using docker containers. FastAPI is used as the web server for populating the frontend UI.

c. Front end

We created a front end that roughly approximates the functioning of an electronic medical record system. The front end has two modes: 1. Patient Explorer and 2. Chart Explorer.

The front end was constructed in this fashion as it mirrors how an EMR is used in practice. Patient information is typically accessed one patient at a time. Patients are seen sequentially during a day in the office or during rounds. At the start of a patient interaction, their chart must be found. The Patient Explorer allows us to search through all of the charts based on name or ID number to find the patient of interest's chart.

Once the patient of interest's chart is found, the Chart Explorer displays the elements of the single chart. These elements include patient interaction notes, scans, x-rays, among others. Our search engine allows the user to query over these elements.

2. Documentation of how the software is implemented

a. Back End

More concretely, we use elasticsearch to create our search engine. We construct three elastic search node clusters in Docker containers. Elasticsearch is used to index our 10,000 chart sample dataset. The code used for creating the inverted index is in ./elasticsearch/indexing.py. After all patient charts are indexed, a web server written in FastAPI is spinned up which talks with Elasticsearch to fetch charts based on searching query. The whole backend stack is containerized via Docker for easy deployment.

b. Front End

The front end is built on Angular running on NodeJs. We use .css files to do the styling. The connection to the back is a hard coded URL. The front end takes a user input and makes a request to the back end API. The response is processed into an array of patient class objects we made.

i. Page 1: A table displaying the important elements of all patients in the response so the user can find the record they wanted.
ii. Page 2: A more detailed view of all the elements of the patient the user selected.

3a. Documentation of how to install and use the software

The README.me file in the root directory of our github repository details the instillation of the software. We have installed our system on Windows under WSL, on Linux, and on MAC. Docker and docker desktop must be installed on your machine.

https://www.docker.com.

a. Clone our repository

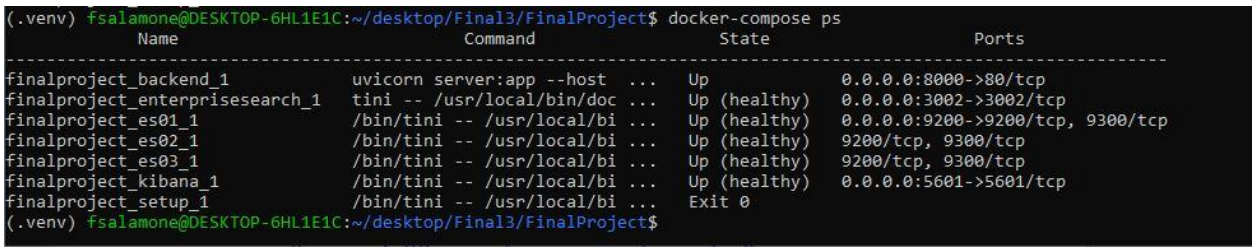
git clone https://github.com/FrankNSalamone/FinalProject.git

b. Spin up elasticsearch containers

Once the repository is cloned, navigate to the root directory and spin up the elasticsearch containers. The sysctl command increases the virtual memory available for elasticsearch.

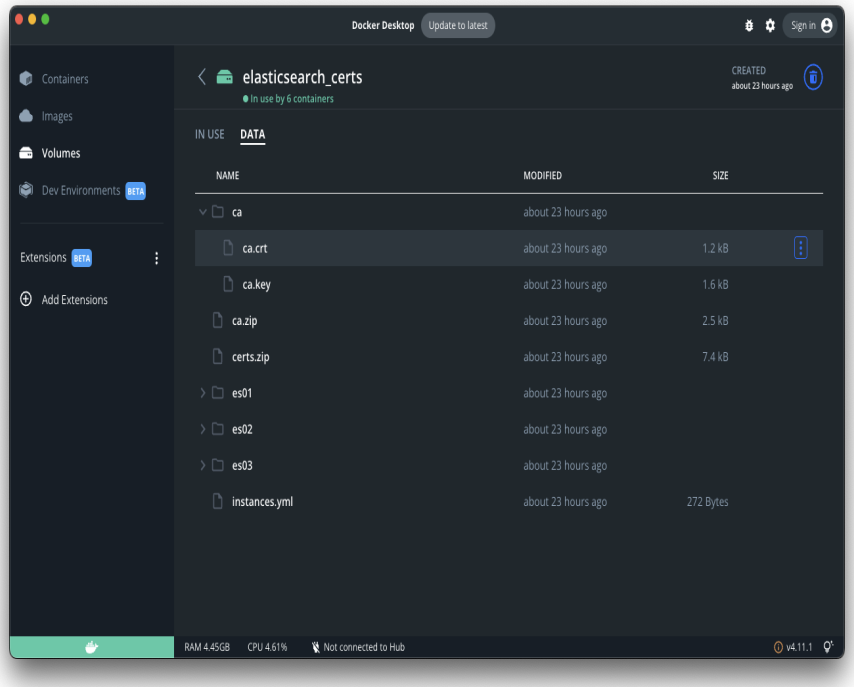
```
# from main directory
# spin up containers
sudo sysctl -w vm.max_map_count=262144
docker-compose up -d
# check whether all containers are running
docker-compose ps
```

Upon running docker-compose, you should see:



c. Make sure certificates are correct

Next, you must replace the elasticsearch certificate in the elasticsearch directory (./elasticsearch/ca.crt) with the certificate obtained from Docker Desktop. Also replace the certificate in ./backend/certs/ca.



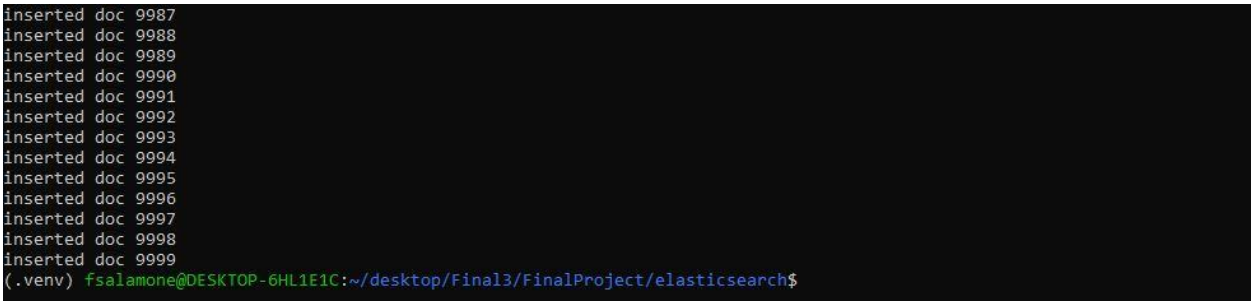
d. Download the patient data (charts2.csv) from the link below and place it in the root directory.

https://drive.google.com/file/d/1iyS_CTQF1S53CmTD1pYua6-aBR2NRtvv/view?usp=sharing

e. Index the patient charts:

```
# go to the root dir of this project first
# activate virtual env
python3 -m venv .venv
source .venv/bin/activate
# install dependencies
pip install -r requirements.txt
# indexing
cd elasticsearch
python indexing.py
```

When this is done, it should look like the image below. Indexing will not work if the certificates have not been copied from docker to the directories on your computer.



f. You may search the charts2.csv dataset by following the link below. You must select our index and then create a search engine. You will have to log in with username: elastic and password: elasticsearch.

http://localhost:5601/app/enterprise_search/content/search_indices

g. You can run the python text client to search patient charts without a GUI if you would like.

```
python text_client.py
```

h. In order to run the graphical front end, be sure that nodejs version18.12.0 is installed (https://nodejs.org/en/download/). You can install npm after you have node installed by running the following inside of the /frontend folder in the project:

```
npm install
```

You can verify node and npm are installed:

```
node -v
npm -v
```

i. Install the Angular CLI so you can run the frontend locally:

```
npm install -g @angular/cli
```

j. Go to the frontend directory inside the project and use the following to start the frontend:

```
ng serve
```

k. When you are done use, ctrl+C to kill it. The frontend should now appear on any browser at http://localhost:4200/. Development IDEs like visual code should be able to

connect to it for debugging and development. (saving reloads the page, and errors show up in the IDE)

l. If you want to run the backend and frontend locally you need to change the hard coded URL in the frontend file frontend\src\app\patient-search\patient-search.component.ts to point to your backend.

3b. Use of our software running on AWS

Installation of our software as detailed in 3a. Above may be difficult because of the environment on your particular computer, so we have made it available on an Amazon EC2 instance at IP address 18.118.170.215. The elasticsearch search engine can be accessed at http://18.118.170.215:5601. The username is elastic and the password is elasticsearch. Our patient search webpage can be found at http://18.118.170.215:4200/search.

4. Team member contributions

Connor Brown (connorb8)

Front end development. Documentation. Look and feel.

Ray Butler (raylb2)

Front end setup and development. Documentation. Look and feel.

Xinlei Huang (xhuang84)

Search engine and back end implementation. Documentation.

Frank Salamone (frankns2)

Web scraping and artificial chart creation. AWS instance. Documentation.

Jonathan Trusheim (jt38)

Back end – front end communication. Documentation. Data formatting.