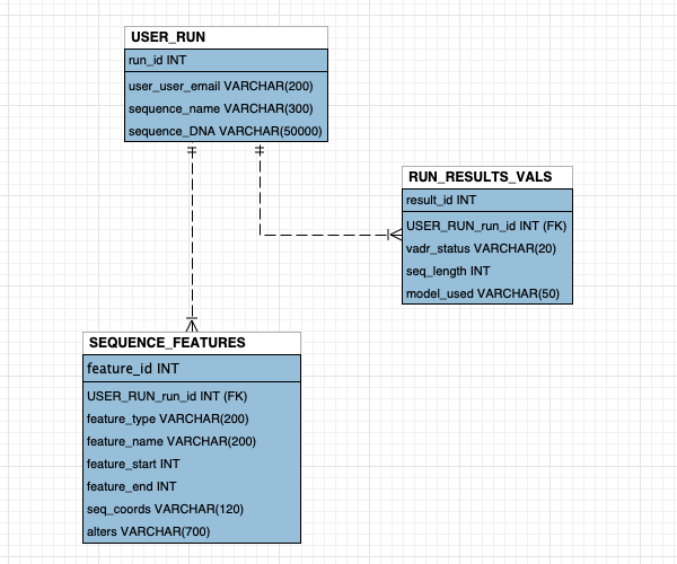
410.712.81 Mauro Chavez
Final Project Write-Up mchavez8@jh.edu

Devastator*: A VADR web app utilizing AWS deployed microservice for sequence analysis

**Darth Vader's spaceship was called the Devastator*

As described in my project proposal, I chose to implement a web app for running VADR (Viral Annotation DefineR), a software package for classifying and analyzing viral sequences against sets of reference genomes [Schäffer 2019]. This tool is important because it serves as an institutional check that determines whether or not a SARS-CoV-2 viral sequence is accepted into the GenBank database. My web app was implemented with the intention of only validating SARS-CoV-2 sequences although I haven't tested it with other viral genomes. One downside of going to a web app for sequence validation is that the viral models used to build the web app might become out of date as new reference sequences are published. If I were to continue working on this project, I'd implement a CICD pipeline so that the analysis module of the web app is redeployed with each new VADR release.

To be transparent, I'd like to address some features described in my initial project proposal that are absent from my final submission. First, there is no real user registration/tracking system or functionality to email users once their analysis runs are complete. For the purposes of this project, a user's email address serves to keep track of their runs and it is a cheap way of identifying who is using the app by having them enter their email prior to run submission and to look up their past runs. Additionally, VADR analysis is so quick that having users look at a countdown timer was much easier than implementing an email system. Second, the database schema was changed slightly. Initially I had two catch all key:value stores for the results; one for single data elements and one for data elements grouped in arrays. These were replaced with more specific tables, one table where rows store metadata for the submitted sequence and another that has a row for each annotated feature of a sequence. The column names were determined based off the data coming straight out of VADR results files. I figured that as there won't be sustained development on this app prior to this class, taking the time to implement a highly flexible and generic schema wasn't worth it. The designated user table was also scrapped. Instead, an email address field was added to the user_run table to store who owns each run. Additionally, as runs complete quickly (usually 10-ish seconds) and polling the microservice for run status is done via a button in the UI, it didn't seem critical to keep track of when a run was initiated and finished so these columns were scrapped form the run table. Third, there is no functionality to display trends/meta data to summarize all the sequences submitted to the app. However, this was initially proposed as a stretch goal to begin with.



The diagram to the right describes the final schema implemented for this project…

Here's what I did instead of the above features: wrap the VADR software in a containerized REST server (written in Python with FLASK) so that it can be easily deployed as a microservice on AWS (github link). When the Devastator web app runs VADR, it's actually

submitting sequences via POST requests to an AWS instance that is running a microVADR server. The microVADR server itself has a small database with a single table that is used to track submitted sequences, their run id, and their analysis status. A record is created when a sequence is submitted to map a run id to an expected output directory. When the status of a run is checked, the run id is used to look up the output directory and search for output files used to populate the return payload. A run is completed when the expected output files exist. Both the microVADR server and accompanying database can be run via a single docker-compose command on the docker-compose file within the repo. The server itself exposes three endpoints; one for viewing all the sequences submitted to the server, one for submitting a sequence, and one for getting the results of analysis. Submitting a sequence results in a response that has a unique run id. To get the results of a sequence, that run id is used in the vadr/{run_id} endpoint. These run ids are guaranteed to be unique based off how microVADR is implemented so they become primary keys of records in the USER_RUN table. A swagger interface for viewing these endpoints and example payloads is available here: http://13.56.20.142:5001/. Writing this service took a significant amount of time but I am glad to have been given the opportunity to investigate AWS, docker, FLASK, and microservices. Please reach out to me if there is an issue with the web app when you are grading as I am not sure how stable my AWS instance is. At one point I left it alone for a few days and it wasn't working when I came back to it. Deploying the microVADR server to AWS was a very simple process due to the service being containerized via docker. Deployment involved installing docker, git, and docker-compose on my AWS instance, cloning my repo, and running `docker-compose up` on my docker-compose file.

One complication I ran into was triggering a cgi script that would submit a run and wait for the run to complete before printing a json payload. No matter what I tried, I couldn't get the ajax request in javascript to wait for the python script to finish before trying to process the expected json output. This is why submitting a sequence in the UI results in a countdown timer that then exposes a button to check for results. This has the unintended consequence of decreasing the amount of times the web app pings the micro service which ultimately is a plus because the micro service is very light weight and might not perform well to many repeated GET requests.

For my test data, the passing sequence is the first published SARS-CoV-2 genome. The failing sequence is one where I added a bunch of random nucleotides which resulted in peptide translation problems, unexpected length, and CDS stop codon/frameshift errors. These issues appear in the ui as sequence feature alerts. The passing sequence has similar annotations but without VADR alerts as the features look legitimate.

Sources:

Schäffer, Alejandro A, et al. "VADR: Validation and Annotation of Virus Sequence Submissions to GenBank." 2019, doi:10.1101/852657.