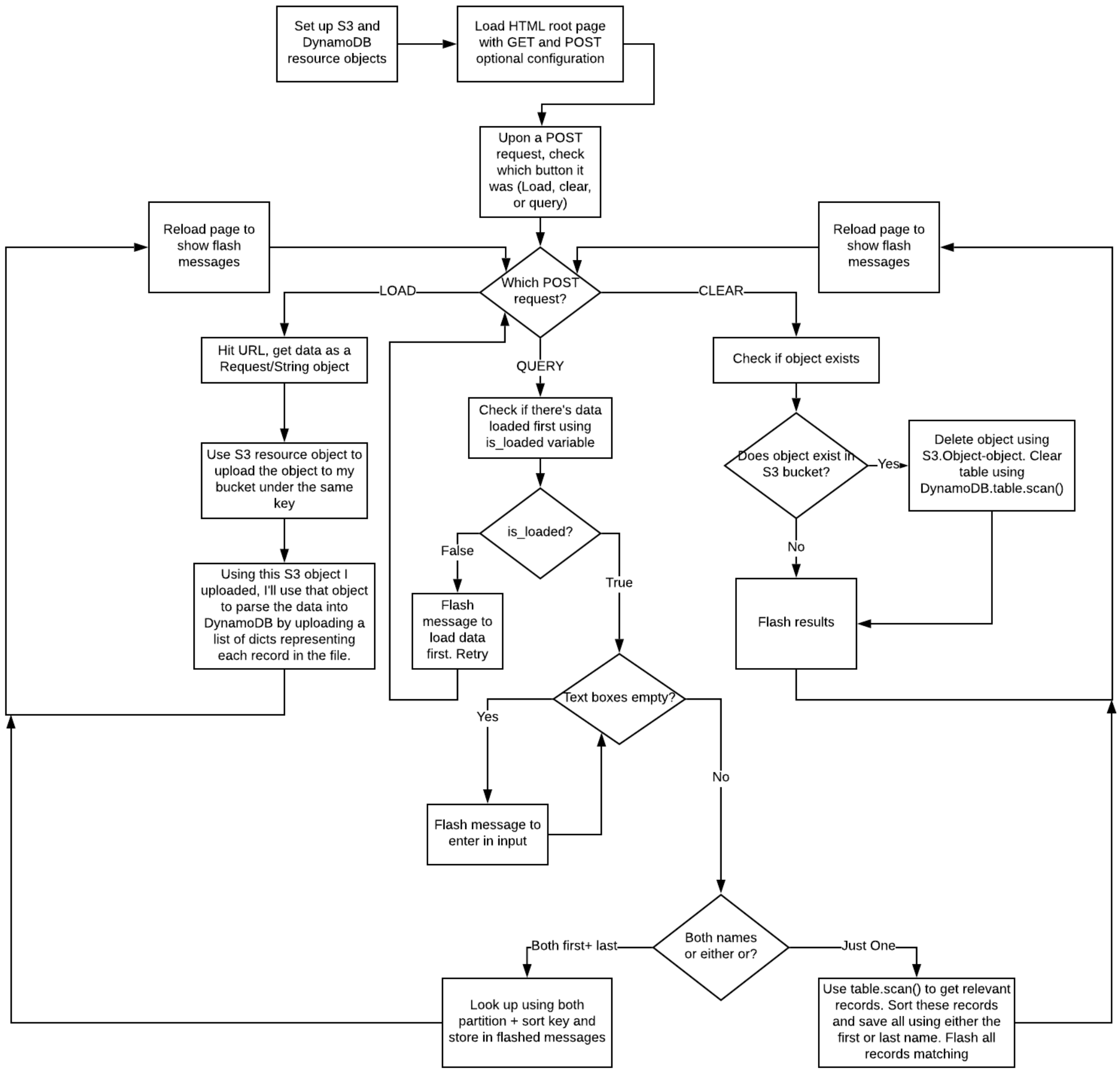
Ji Kang
Program 4 Report
CSS490
Prof. Dimpsey

Site URL: **http://jikangcss490.us-west-2.elasticbeanstalk.com/**
S3 Object URL: **https://s3-us-west-2.amazonaws.com/jikangprog4bucket/userDB.txt**
If the object is not there, it's been cleared before I submitted. Please hit the website and press 'Load Data' to load it into the bucket.

Diagram of flow of the program


Set up S3 and DynamoDB resource objects
Load HTML root page with GET and POST optional configuration
Upon a POST request, check which button it was (Load, clear, or query)
Reload page to show flash messages
Which POST request?
LOAD
CLEAR
QUERY
Reload page to show flash messages
Hit URL, get data as a Request/String object
Use S3 resource object to upload the object to my bucket under the same key
Using this S3 object I uploaded, I'll use that object to parse the data into DynamoDB by uploading a list of dicts representing each record in the file.
Check if there's data loaded first using is_loaded variable
is_loaded?
False
True
Flash message to load data first. Retry
Text boxes empty?
Yes
No
Flash message to enter in input
Both names or either or?
Both first+ last
Just One
Look up using both partition + sort key and store in flashed messages
Use table.scan() to get relevant records. Sort these records and save all using either the first or last name. Flash all records matching
Check if object exists
Does object exist in S3 bucket?
Yes
No
Delete object using S3.Object-object. Clear table using DynamoDB.table.scan()
Flash results

Explanation of Flask's dependencies
Flask apps have a requirements.txt file which lists the dependencies needed to run the app. That is what 'requirements.txt' is for. I'm also utilizing two standard HTML templates to represent the hope page. One is a Jinja2 template and the other is a standard HTML file which utilizes the Jinja2 template to print out the messages done via flask(…). Flask routes URL resources to different application.route(…) which represents pages. The templates are located in the templates folder. Application.py is the program to run.
Elastic Beanstalk's default configuration has the following requirements:
Program to run is named 'application.py' with the flask object being named 'application'.

Site scaling with load
I'm utilizing the free-tier of Elastic Beanstalk's configuration which only utilizes a t2.micro instance which is one of the lowest IO ops/second offerings on the service for my EC2 instance. With the default configuration, it only comes with 100 IOPS for this provisioned SSD volume. This may be a low IOPS for a standard stand-alone webapp but I've added a default elastic load balancer which has a SLA of 4 9s itself. It doesn't utilize cross-AZ load balancing but will help the site scale with load. If the given site's IOPS exceeds the given 100 IOPS limit for this t2.micro instance, it can provision another server to handle the overflow. I've set this up to go up to 3 total with a single elastic load balancer. This would mean my web-app would be able to handle up to 300 IOPS before services started to slow down for the users.

Monitoring done on the site
For the beanstalk service, it utilizes Amazon's CloudWatch which is currently configured to look at my service every minute. It'll get information such as CPU utilization, latency, total requests in that period (which'll be used to scale the website with the load balancer), Network in vs. Network out and environment health. CloudWatch will ping the site, look at the response code, and log the information which'll be sent to me either through email or text.

Estimation of SLA
The current metrics are…
- EC2 SLA = 4 9s
- S3 SLA = 4 9s
- DynamoDB SLA = 4 9s

My program will hit the S3 object provided for us and is dependent on that data to run = 0.9999

Storing that S3 object's data into my own S3 object in my bucket, my program reads off of the S3 object I've stored which is another 0.9999.

For querying, it relies on DynamoDB to be up and functional which is another 0.9999.

Load balancer has a SLA of 4 9s also but I only need one instance to be functional even if the instance may slow down due to an overload. A single EC2 instance has a SLA of 0.9999. Only need one of the 3 to be available. 1 – (failure rate ^ 3) = 1 – (0.0001^3) = 0.99989999999.

So the SLA for the service, just to maintain availability (may slow down), is…
(0.9999^3) * (0.99989999999) = **0.99960005997 or ~99.96%**