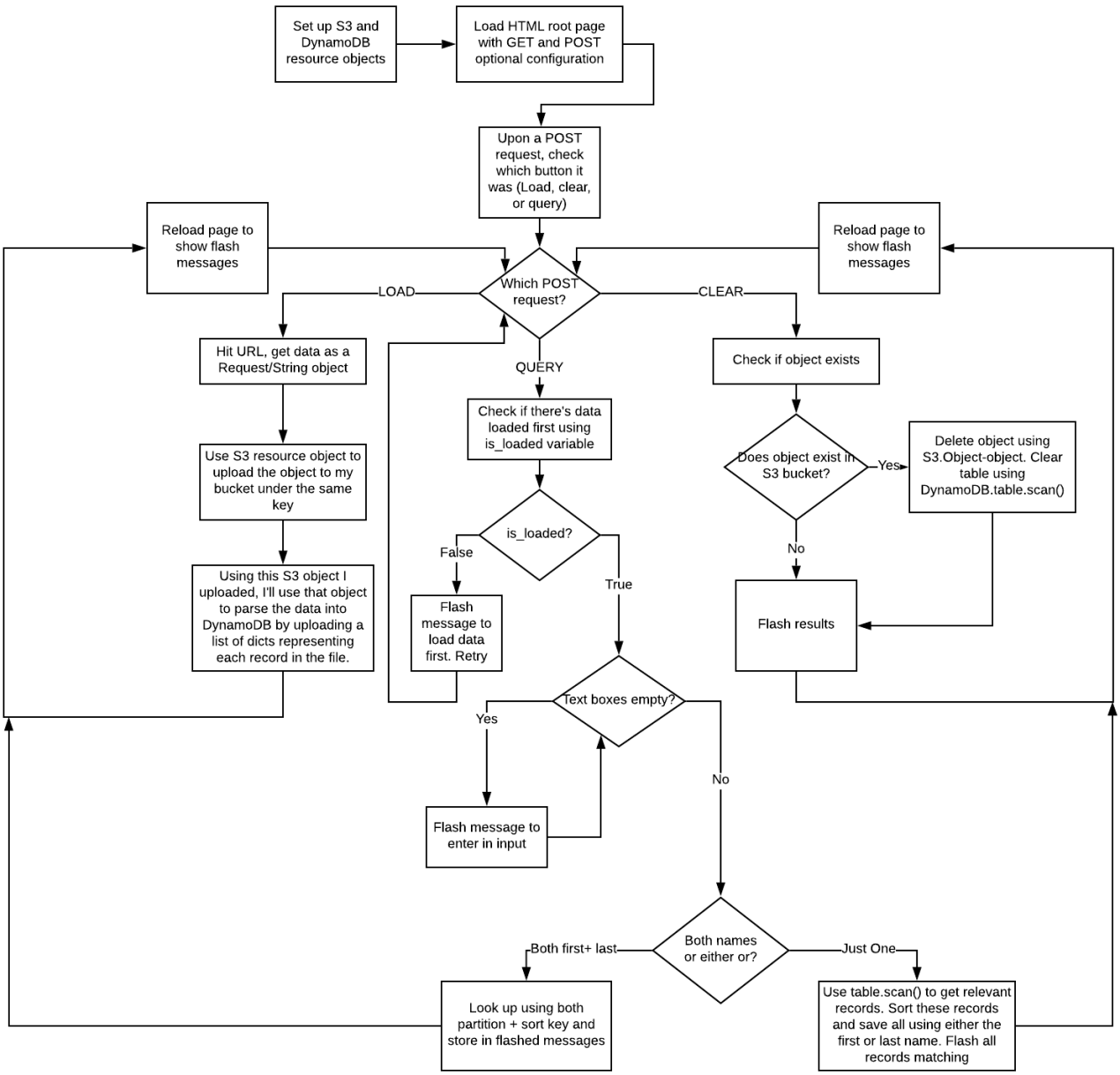
Ji Kang
Program 4 Report
CSS490
Prof. Dimpsey

Site URL: **http://jikangcss490.us-west-2.elasticbeanstalk.com/**
S3 Object URL: **https://s3-us-west-2.amazonaws.com/jikangprog4bucket/userDB.txt**
Utilizes the AWS S3 object URL provided.
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



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'.

Program overall
Bare HTML page is linked to the flask backend using request.form which can get the names of the given inputs and map them to names which allows me to see which button is being pressed.
Based upon that, I do simple checks. Can't query without loading data first. Can't clear what's not there, and so on.
For loading data, it'll hit the website provided with retry logic implemented. It'll get the input and upload as a object to S3. This newly uploaded S3 object is then hit and the text input is parsed as a list of dicts. DynamoDB offers a very nice upload_object method which'll accept dicts which have key mappings to the primary key(s) in your table.
For clearing data, it'll essentially check if there is any data to be cleared. If so, it'll delete the object using a S3.Object.delete() method and clear the DynamoDB by getting all primary keys of all records using the Table.scan() method. I've read that clearing a table all together is much faster but creating a table may have a very high overhead (1-3 minutes). For that reason, deleting just the records seem to be a much better use of time.
For querying data, It'll check if data is already loaded, first+last name, just first, and just last name. Based upon this input, it'll perform different things. For first and last, it can just look up the dynamo DB table with the partition and sort key. For just first or just last, it'll need to scan through all records using .scan(). It'll add these records to a list which is then iterated through.

All output is done via Flask's flash(…) method which'll prepend the information before the html page AFTER a reload. As shown on the flow chart diagram. Flash(…) will store all of this information and blurt it out after a reload which is the basis of this program's design.

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%**