**Overall Architecture:**

The overall system being designed is a communication services. It will have three main components that can handle SMS, Email and push requests. In the diagram below we only layout the architecture for the SMS service. It will work closely with the Login Service and could utilize a technology like JWT to allow for handling both security and scaling.

A picture containing text, screenshot, rectangle, diagram

Description automatically generated

Figure 1: Overall Architecture

**SMS Micro Service:**

This service will use a RESTful API that will take incoming requests and send them to our users. It will primarily use AWS SQS for sending but will also have Twilio as a backup. The service will accept requests from other microservices or user devices. There will be two main components of the service.

* A single text service for individual users a single text
* A group texting service for sending bulk text to multiple users

**Logic Flow:**

A producer (user or microservice) will request that an SMS be sent to an individual user or group of users. The Communication Service will handle this pending request and store it in a cache like Redis or AWS Elasticache running Redis. It will also send back a Request ID to the originating service with a **pending** status.

From here the request will be stored in a message queue. Many cloud providers have these available which could be utilized before building one internally. If the request is successful it will return a 200 response and it’s status will become **resolved**.

A picture containing text, screenshot, diagram, rectangle

Description automatically generated

Figure 2: SMS Service

If the request fails it will retry internally at a specific interval for three times. On the third time it will be removed into persistent storage like a SQL database table (sms\_failed\_requests) where we can potentially retry these failed requests at a point in the future. There will also be a service tracking total failed requests at any given time. Should this level reach a threshold we deem important (say 100 failed individual messages or 5 failed group messages we will auto cut a ticket to the team on call supporting this service.

**Architecture:**

* Routes
  + POST: api/text/user
  + POST: api/text/
* Logic
  + Java Class based
    - SingleSMS
    - GroupSMS

**Class Overview:**

SingleSMS

* This class (or multiple classes) will be

**Database:**

Tables (SQL or DynamoDB)

* User Table
* User Device Table
* SMS Failed Requests

Cache (Redis or Elasticache)

* SMS Pending Requests

**Fault Tolerance:**

* The system will be monitored with a service like AWS Cloudwatch with alarms set up to trigger a ticket for certain triggers at a specific threshold. For instance a high level of 500 errors being returned.