

# BUILDING A NEWSLETTER SERVICE WITH PUB-SUB ARCHITECTURE

Nikhil Gangisetty  
School of Science and  
Engineering – Computer Science  
University of Kansas City Missouri  
Kansas City Missouri  
nikhil.gangisetty@umkc.edu

Jagan Kasula  
School of Science and  
Engineering – Computer Science  
University of Kansas City Missouri  
Kansas City Missouri  
jgkghb@umkc.edu

Ben Curtis  
School of Science and  
Engineering – Computer Science  
University of Kansas City Missouri  
Kansas City Missouri  
bcd7c@umsystem.edu

Kalyan Naga Sai Nayani  
School of Science and  
Engineering – Computer Science  
University of Kansas City Missouri  
Kansas City Missouri  
knxy3@umkc.edu

## 1. ABSTRACT

Group 10 has begun developing a newsletter service using Pub-Subscribe Model. A newsletter service is a powerful tool for businesses, organizations, and people looking to create and maintain relationships with their target audience, enhance engagement, generate leads, amplify their message, and receive vital data and insights. In this project, we will use the pub/sub architecture to create a newsletter service. Considering that it is a great method for creating a comprehensive and scalable newsletter service because it offers content customization, guaranteed delivery, loose coupling, and temporal decoupling.

## 2. RELATED WORK

Overall, multiple articles were sourced and identified within the publish/subscription module. We have listed the top 4 from our research below.

The 1<sup>st</sup> identified article “A survey of Publish/Subscribe Middleware Systems for Microservice Communication [3]. This discussed a software architecture that “divides a large application into several little autonomous services. It allows loosely connected services/applications to operate together. The services are deployed independently and communicate via a lightweight remote call.” As noted below, this was the direction taken within the project.

The 2<sup>nd</sup> article was Loupe: Verifying Publish-Subscribe Architectures with a Magnifying Lens [4]. This was an interesting dissection on how to improve timing without the Pub/sub process. Instead of FIFO, it proposed that group messages are to be used to aid in time savings.

In the final article, Building Fault-Tolerant Overlays With Low Node Degrees for Topic-Based Publish/Subscribe [5], it speaks to “new approach for designing reliable and scalable overlay networks to support topic-based pub/sub communication” with “MinAvg-kTCO problem parameterized by k.” The article goes in great depth for the various algorithms used as well as the results obtained.

## 3. PROPOSED TECHNIQUE

Group 10 proposed technique will be to implement a Newsletter service using Publish/Subscription model with UI login for the publisher, subscriber.

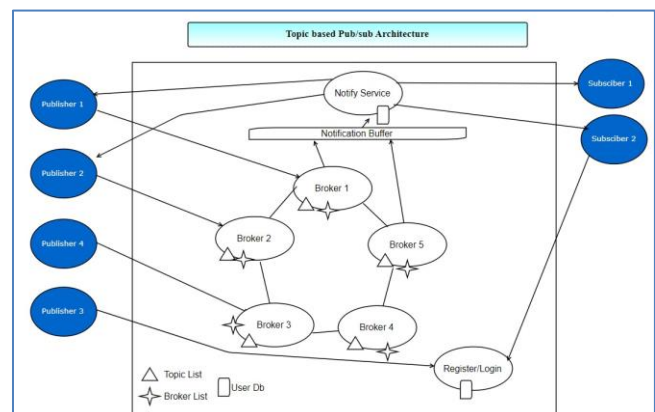


Figure 1: Group 10 Pub/Sub system architecture version 1.

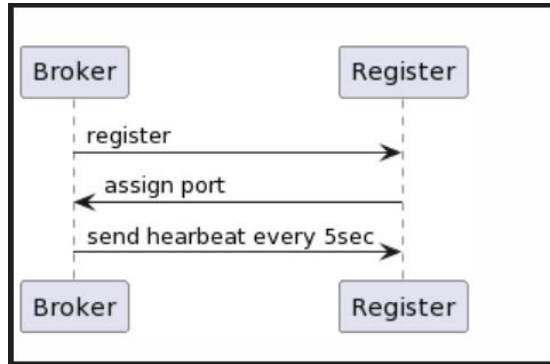


Figure 2: Broker Registration

Publisher:

There will be publishers which will interact with a login system to gain access to the system then communicate with the broker.

Broker:

There will be 5 brokers, each broker handles N/5 topics. They will receive publisher data in the form of articles and tokens then send subscriber content to the notification service.

Register Service:

The broker will register with the registration service and in return be assigned a port. The broker will then send a heartbeat every 5 seconds.

Login service:

The login service will receive request logins from publishers and subscribers via a graphic interface. In return, it will send welcome tokens to them. For subscribers, the service will also receive a request registration and send a registration confirmation in return.

Notification service:

The notification service shall notify via email the subscriber when it is received from the broker. It will also notify the publisher with the number of subscribers that received its content.

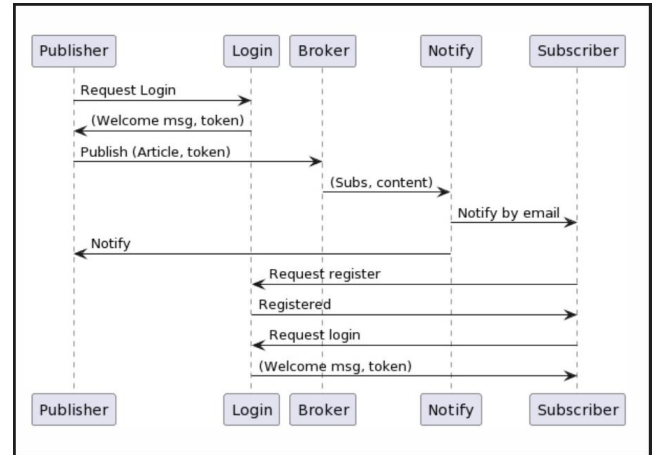


Figure 3: Event and Process Interactions

The project shall be organized in the following:

- Code will be developed in Python.
- Code shall be placed in GitHub repository for organization and tracking.
- GitHub feature of issues will be used for outlining the initial tasks needed to succeed. Labels have been applied to indicate priority, milestones, resource and estimated time for completion.

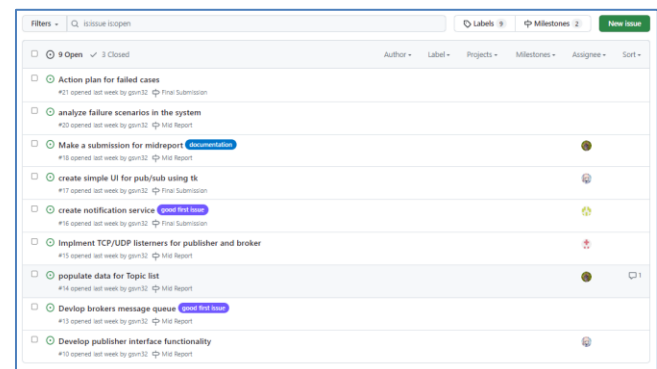


Figure 4: GitHub task list with labels and milestones assigned.

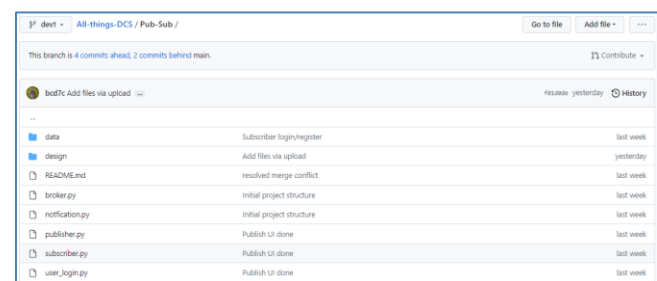


Figure 5: GitHub pub/sub repository with version controls.

## 4. IMPLEMENTATION

During the implementation phase, modifications were made to the original plan as the pieces of the code came together. Key items identified that TCP listener was a better alternative for the final project. A similar decision was made for the brokers in transitioning from a leader election to the brokers having a single role with no leader process.

**Publisher:** This is a Python script that creates a GUI for a publisher client that allows users to log in or register, publish messages to a broker, and receive notifications. It uses the Tkinter module for GUI elements and the Requests and Socket modules for network communication.

The GUI contains three frames: a login page, a registration page, and a main page. The login page has fields for username and password, as well as a login button and a link to the registration page.

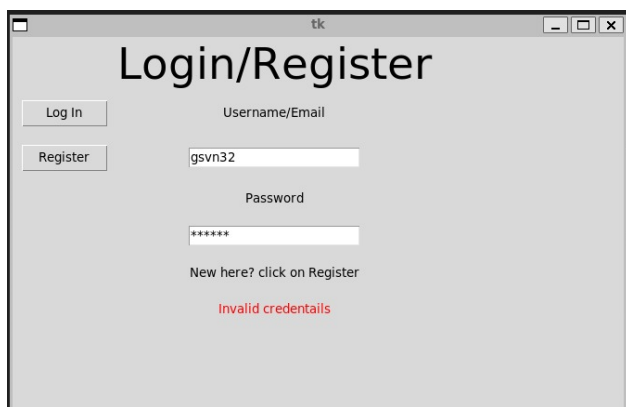


Figure 6: Login/Register User Interface

The registration page has fields for name, email, username, and password, as well as a register button. The main page has fields for topic, title, and content, as well as a publish button and a notifications area.

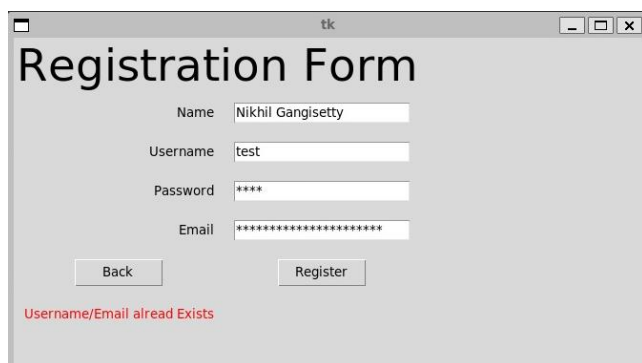


Figure 7: Register Form User Interface

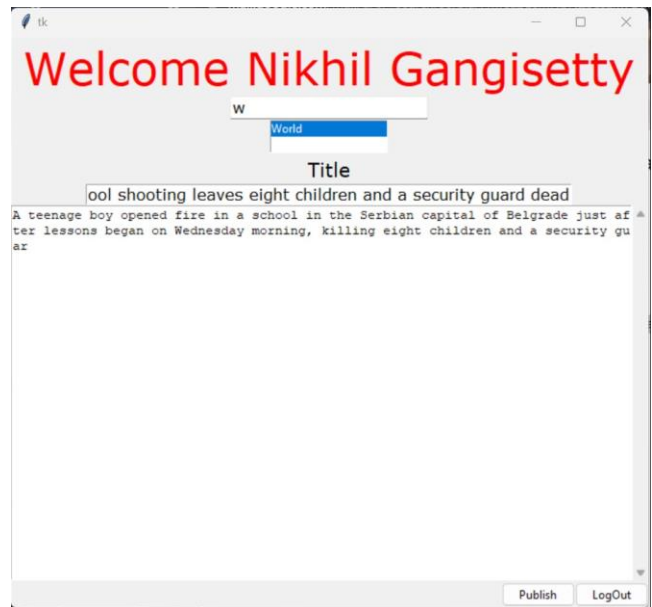


Figure 8: Publisher Main Page

The script also defines functions for retrieving topics and broker information from a database, validating login credentials, and registering new users with an HTTP server, and publishing messages to a broker using TCP sockets.

**broker:** The code is a Python implementation of a message broker system. It starts by importing the necessary modules such as json, threading, queue, socket, and sqlite3. Then it defines some global variables and creates a message queue. It registers the broker to topics by calling the remote `reg_topic()` function using `xmlrpc.client`. The code defines functions to retrieve the subscribers for a given topic, handle incoming messages, and process the messages in the message queue.

It creates a TCP listener thread that listens for incoming messages, a message processing thread that processes the messages in the message queue, and a heartbeat thread that sends a heartbeat signal to the broker every 5 seconds.

When a message is received, it is decoded as a JSON object, put into the message queue, and then processed. The subscribers for the topic are retrieved, and the message is sent to the `notify_service` by creating a TCP connection, creating a JSON object, and sending it to the notify service.

**user login:** This code implements users' registration and login system using HTTP POST requests. The server runs on localhost and port 25678 and listens for incoming requests. The system uses SQLite to create a database with a single table called `users` and columns `uname`, `name`, `pass`, `email`, and

token. The `create_new_user()` function creates a new user record with the specified values, and `get_user_credentials()` retrieves the username and password for a user given their username or email. `check_user_exists()` checks if a username or email already exists in the database. `update_user_token()` updates the token for a user given their username or email.

The `MyServer` class defines the server and implements the POST method handler, which reads the request body, parses it as JSON, and based on the action specified in the request, calls either `action_register()` or `action_login()`. `action_login()` retrieves the user's credentials from the database and generates a token for the users if they are found. `action_register()` checks if the username or email already exists in the database and creates a new user if not. Both `action_login()` and `action_register()` return a JSON response indicating whether the action was successful or not. Finally, the server object is created, and the server is started and listens for incoming requests indefinitely.

**Notification service:** This is a Python script that sets up a message queuing system using threads and sockets. It listens for incoming messages on a specified port, handles them, and puts them in a message queue. Then it processes the messages in the message queue and sends email notifications to the subscribers and publisher of the messages.

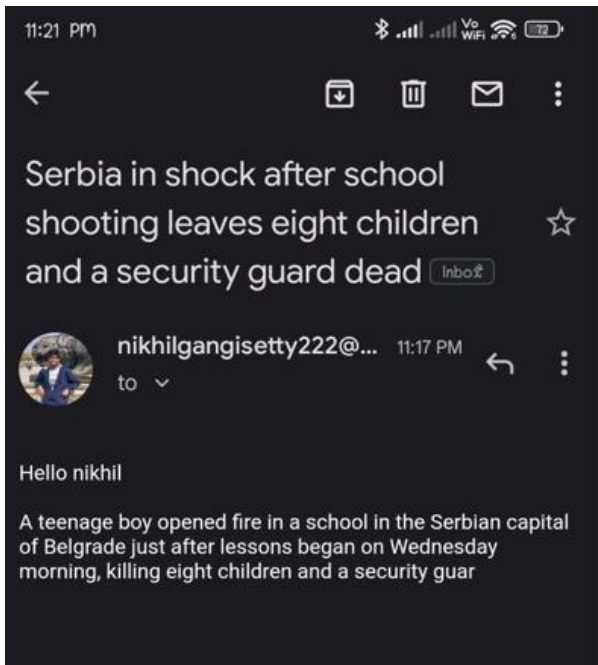


Figure 8: Subscriber Notification



Figure 9: Publisher Notification

The script imports several modules including `json`, `threading`, `queue`, `socket`, `smtplib`, and `sqlite3`. It defines a TCP listener function to handle incoming messages, a message processing function to process the messages in the message queue, a function to retrieve the name and email of users given their username, and a function to send emails.

The script creates a message queue named `'my_q'` and starts two threads - one for the TCP listener function and the other for the message processing function. The TCP listener thread listens for incoming messages on a specified port, handles them, and puts them in the message queue. The message processing thread processes the messages in the message queue and sends email notifications to the subscribers and publisher of the messages.

The script uses `SQLite` to connect to a database and retrieve the name and email of a user given their username. It also uses the `smtplib` module to send email notifications to the subscribers and publisher of the messages.

**register:** The code creates an XML-RPC server to handle broker registration and heartbeat functions. It uses `SQLite` to store information about brokers and their assigned topics. The server assigns a free port to the brokers and keeps track of active ports using a separate thread. The server listens on port 8000, and broker ports range from 8001 to 8005. The server has two functions: `reg_topic` to register brokers and assign topics to them, and `reg_heartbeat` to keep track of active brokers.

## 5. AUTHOR CONTRIBUTIONS

Contributions towards the Term project have occurred in various ways. The primary communication channel has been the WhatsApp app. This has allowed the sharing of ideas, topics, and concerns as the project started to progress. Discussions with the focus on requirements and a system architecture to aid in avoiding require and ensure that all members of the group could have a solid understanding of key concepts that would then allow us to use a divide and conquer method for completions of the project. We have also had multifaceted discussions that have allowed us to being coding foundational modules. A GitHub repository has been established to aid in managing, documenting, and organizing the projects' progress. As each team member has various degree of programming knowledge from advanced to none, we must be careful that each piece is provided and aligned with the team members current skill set.

Current team members are Nikhil Gangisetty, Jagan Kasula, Ben Curtis, and Kalyan Naga Sai Nayani

## 6. REFERENCES

- [1] S. Vinoski, "Advanced Message Queuing Protocol," in IEEE Internet Computing, vol. 10, no. 6, pp. 87-89, Nov.-Dec. 2006, doi: 10.1109/MIC.2006.116.
- [2] Coulouris, G., Dollimore, J., Kindberg, T., & Blair, G. 2013. Distributed Systems: Concepts and Design (5th ed.). Pearson Education (US). DOI: <https://umkc.vitalsource.com/books/9780133464917>
- [3] S. Kul and A. Sayar, "A Survey of Publish/Subscribe Middleware Systems for Microservice Communication," 2021 5th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT), Ankara, Turkey, 2021, pp. 781-785, doi: 10.1109/ISMSIT52890.2021.9604746.
- [4] L. Baresi, C. Ghezzi and L. Mottola, "Loupe: Verifying Publish-Subscribe Architectures with a Magnifying Lens," in IEEE Transactions on Software Engineering, vol. 37, no. 2, pp. 228-246, March-April 2011, doi: 10.1109/TSE.2010.39.
- [5] C. Chen, R. Vitenberg and H. -A. Jacobsen, "Building Fault-Tolerant Overlays With Low Node Degrees for Topic-Based Publish/Subscribe," in IEEE Transactions on Dependable and Secure Computing, vol. 19, no. 5, pp. 3011-3023, 1 Sept.-Oct. 2022, doi: 10.1109/TDSC.2021.3080281.