TAC WORLD

SOFTWARE DOCUMENTATION

Instructions for running the server and client respectively

- Setup Docker
- Clone the Repo: https://github.com/Midhun-Chandrasekhar/tac-world.git
- Navigate to the repository
- Execute: docker-compose up --build --scale app=2
- Entire Stack with 1 Loadbalancer, 2 App server, 1 Redis server, and 1 MongoDB will start
- Navigate to http://localhost:81/ and use the App
- To Load Test:
    - Install libraries using: pip3 install requirements.txt
    - Run Test: python3 tac/test/test_chat.py

Technical choices and Trade-offs

Front End /UI Application

HTML

Alternative: Mobile FE Apps, Desktop FE App
Reason: HTML can be rendered in any platform which supports the web browser. Most of the Mobiles, Laptops, Tabs, Desktops, etc support web browsers. HTML Apps are small in size in comparison with other platforms. Also, HTML Apps are less costly to maintain.

CSS
Alternative: Material UI, Bootstrap, SemanticUI
Reason: The Application scope is minimal. Other choices were not minimal. Implementing them will increase my UI load and chances are high for reduced performance. CSS has native support with web browsers. And using it, we can create an element that is necessary for our App.
It also helps to maintain minimalism.

JS
Alternative: None to replace but have some superscripts like Typescript
Reason: The application scope is minimal. It only contains only 3 views. JavaScript is more than enough to handle all the required cases. But if a wider scope arrives I would love to opt for React + Typescript combo.

Back-End Application

Python
Alternative: Javascript, Golang
Reason: Python is a general-purpose programing language and has huge developer support and library base. The objectives of the project are easily achievable by python and It's my primary stack. I also had a quick reading on Golang. I believe Golang can offer more performance compared to python. But time constraints matter. Proper architecture and clean code can make things easy with python for this use case. Javascript was not mentioned in the requirements.

Flask
Alternative: Django
Reason: Flask is a micro web framework for building microservices using python. Many AI application engines are served by the flask. Even some services of Netflix are powered by the flask. Django is a full-stack web framework. It will make things heavy here. So it's out of my list.

Flask-SocketIO
Alternative: Django Channels
Reason: Socketio is a great library to build a TCP-based real-time chat application.
Flask-SocketIO is a python wrapper on Socketio and it's designed in a way that the interaction between flask and Socketio is easier. If the platform was Django, I may have opted for Django channels. Since my choice of web framework was Flask, I chose Flask-SocketIO.

MongoDB - NoSQL
Alternative: SQL
Reason: The Entire story of the application revolves around Messages and Users. And it's sensible that the Message is the one who is going to conquer the majority of the space. Messages have an unstructured design. It can be anything, a text, an image, audio, and so on. Keeping such data in Tables is not ideal. This is the primary reason for choosing NoSQL DB. MongoDB is pretty popular, scalable and performant, and affordable. The trade is with the relationship between collections inside the DB. NoSQL also reduces the migration overhead.

MongoEngine
Alternative: Pymongo
Reason: MongoEngine is a popular ODM. It will help us to mimic the object property over the documents of NoSQL DB. Moreover, MongonEngine is capable of maintaining connectivity with DB by introducing necessary techniques like pooling. This reduces the overhead of handling low-level DB communication use cases. Moreover, it can even help us to place relationship constraints and data structure constraints over the Document. Even though there is no DB level constraint, ODM can help implement it from the application level. Pymongo is like a connector, it doesn't have any ODM features.

Redis
Alternative: RabbitMQ
Reason: Redis is an open-source in-memory data source that can function as a message-broker, database, and cache. It supports various data structures such as strings, hashes, lists, sets, etc. It is quite fast and lightweight compared to RabbitMQ. Large messages are better handled by RabbitMq and are a tradeoff here. Redis can handle millions of messages and is very scalable for our use case.

Gunicorn
Alternative: UWSGI
Reason: Gunicorn is a lightweight Python HTTP server. It's popular in deploying projects based on python like Django application, Flask app, etc. According to trends, Gunicorn has higher popularity in comparison with UWSGI. Moreover, Gunicorn is built from the model of the Unicorn project. UWSGI is fairly complex in comparison with Gunicorn. But as a tradeoff, UWSGI marks its fast performance.

Eventlet
Alternative: Gevent
Reason: Gunicorn is not good at handling concurrent tasks. To make it capable wee need to attach a worker to it. Eventlet is a popular one for that task and is stable. It's a concurrent networking library for python and is highly scalable regarding non-blocking I/O. Eventlet is having active development and is showing an uptrend. Gevent is in its matured sate is showing a

downward trend in growth. Gevent’s popularity is the tradeoff here. Anyways, Eventlet is not that complicated.

Docker
Alternative: Bare metal
Reason: Modern applications are developed as containers. This makes them highly portable and easy to ship. Also, they are economically efficient. We can make use of the physical infrastructure effectively using dockers. Even the deployment process is faster in comparison with bare-metal resources. Bare-metal are fairly easy to set up in comparison with dockers, I mark it as a tradeoff here.

Nginx
Alternative: Apache
Reason: Nginx is widely used as a load balancer by many applications. Nginx is a webserver is having non-blocking I/O implementation in it. This made it my choice for here. Even though Apache is highly popular, I prefer Nginx over Apache here.

Think Ahead

New Features
From the current state of the application, there is a huge arena of opportunities are available for this application to rise up. Currently, the application only supports Text messaging. Real-time video conference, Realtime Audio conference, Real-time gaming, Audi chats, image chats, stickers, challenges tasks and etc can be implemented over the platform. This can make users more engaging in the perspective of the application. Moreover, we have to be more focused so that the application ends up like traditional competitors. We have to focus on the technical enthusiast area, in short, the application features should engaging for them too. Usage of AI, to notify the users of the latest technology and creating an environment in which the user can make conversation with our AI bot to understand more about the subject is also a potential story point. We have to deliver multimedia messaging first. Then we have to focus on real-time multimedia communication. Then on Real-time gaming and then to the scopes of AI as discussed above.

Global Audience
As a real-time chat application, the scope of a global presence has a trivial role. As per reports, stack overflow has around 14 million users registered in their system. This means a platform like this has more acceptance space in the global market. But the application needs to go through valuable changes and updates that can make it ready to extend its presence around the world.
From the perspective of a client, the application will be more acceptable if it supports the preferred language of the user. Also, it found that the users from different countries have different UI/UX trends. So a dynamic UI that is capable to make self-updates can help the system a lot or at least a FE architecture that can incorporate dynamic/quick changes.
Regarding BE, We need to focus on multi-region deployment strategies. The availability of the application and its performance are important. Proper infrastructure estimation, Backend logic to adapt the different users from different regions, Data analytics, Database migrations, and so on. As a Road map, I prefer to implement the multi-region support and different language support inside of our system. Covering region by region makes it easy. So choose a region to develop changes specific for them and then plan to do a deployment to that region. Always find a generic architecture in each stage of deployment. Gradually this can task of expansion will become less effort task.

Security
Currently, we are using just the username for logging into the system. This API itself is going to cause the system a lot of troubles like bot attacks, DDOS, etc. So in terms of security, I would like to implement a proper authentication and authorization mechanism on this application first. Once it's completed I would love to extend it to two-factor authentication. This can ensure security inside the application. Regarding the application, First, we need to update the Nginx conf with proper security measures and performance measures. Also at the application level, we need to ensure that the ODM is not exploitable by SQL attacks. Also, proper CORS has to be implemented. Only known entities should have access to our system. We need to properly configure the infrastructure. I intend to place a WAF + Application Load Balancer upfront in the public network. The application server, Database, and message queue should be placed in the private network. TLS/SSL security should be implemented in the load balancer. We also need to implement a proper logging mechanism for our application. Also, an Error report mechanism will be useful for the system to identify the issues at the earliest. To and fro, the checklist keeps on increasing. As we grow the requirement for security also keeps on growing. We should always keep an eye on issues around because the application has a real potential to become a part of the day-to-day life of some of the valuable human resources around the world. Identity theft is always a valuable movement in the phishing industry. Even if the client is not aware of an application we should enforce security. This can help the application to have higher credibility among its competitors.