Distributed and Parallel System
Midterm Report

Group Member:

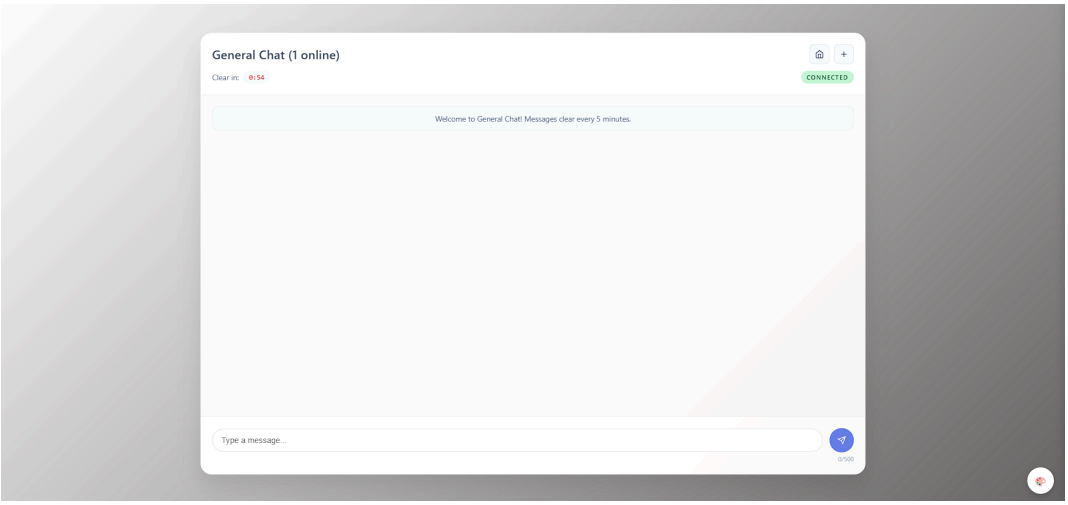
Group Member I	: Ernest Teo	001202300142
Group Member II	: Willy Tanuwijaya	001202300208

Cikarang, Bekasi, Indonesia
Monday, July 10th, 2025

PRESIDENT UNIVERSITY
2025

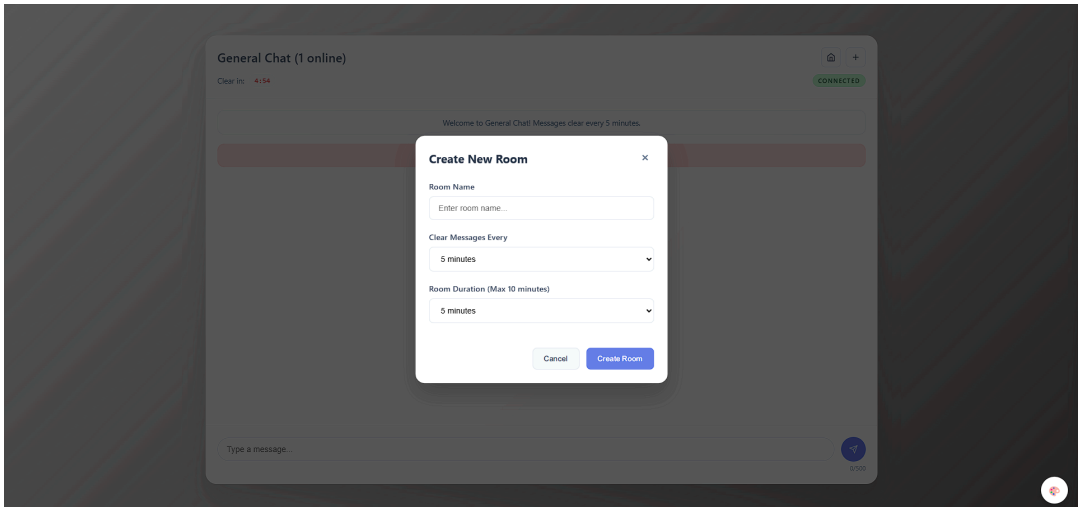
Anonymous Realtime Livechat

- is a web-based anonymous live chat system designed for instant and private interaction without the need for usernames or logins. Users are immediately directed to a public General Chat room where messages are automatically cleared every 5 minutes to prevent server overload and ensure smooth performance. For more private conversations, users can create custom rooms with configurable message clear intervals and timed expiration. The system also supports real-time user tracking per room and allows users to switch between chat rooms easily. Additionally, a theme customization feature lets users personalize the chat experience by changing the background color.



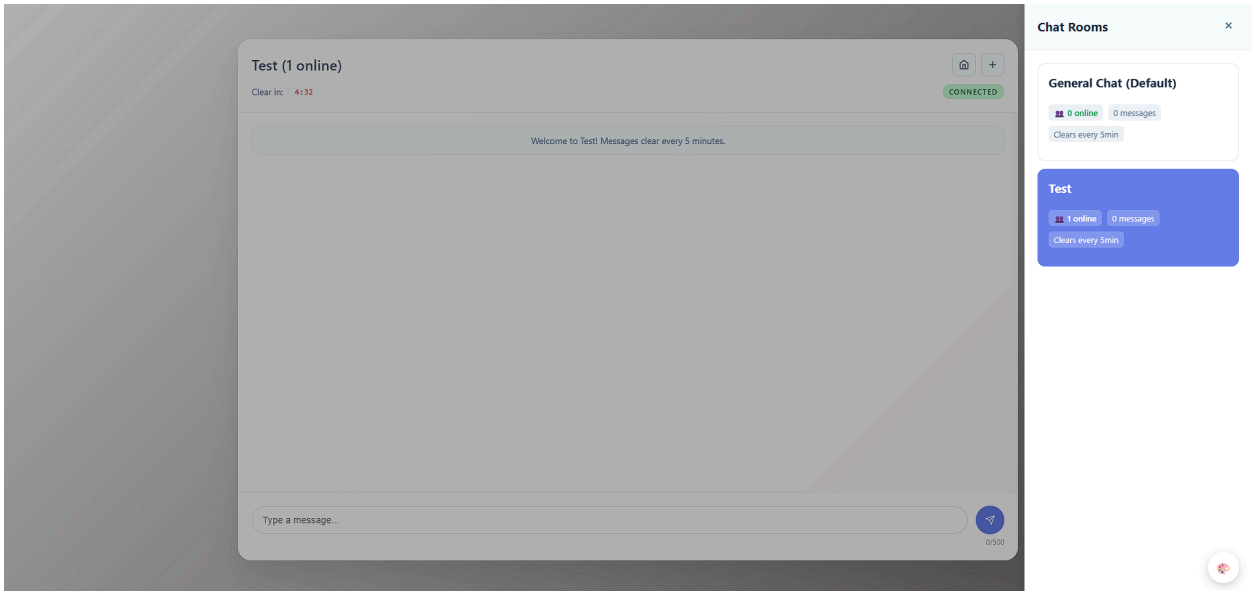
General Chat Page:

The default landing page is an open anonymous chat room called “General” where everyone can chat freely. Messages here are cleared every 5 minutes automatically.



Room Creation Menu:

Users can create custom rooms with options to set the message clear interval (1, 2, 3, 5, or 10 minutes) and a room expiration timer (2, 5, or 10 minutes) to auto-delete the room after inactivity.



Room Joining & Listing:

A sidebar menu allows users to view active rooms, see how many users are currently in each, and join any available room instantly.

Dockerfile (backend)

```
FROM python:3.11-slim

WORKDIR /app

# Copy requirements first for better caching
COPY requirements.txt .

# Install Python dependencies
RUN pip install --no-cache-dir -r requirements.txt

# Copy application code
COPY . .

# Expose port
EXPOSE 5000

# Run the application
CMD ["python", "app.py"]
```

FROM python:3.11-slim
This line specifies the base image for the container. It uses the official Python 3.11 image in its "slim" version, which is a lightweight variant that reduces the image size and download time. It includes the Python interpreter but excludes unnecessary tools and packages.

WORKDIR /app
This sets the working directory inside the container to /app. All subsequent commands (such as COPY, RUN, and CMD) will be executed from this directory. If the directory does not exist, it will be created automatically.

COPY requirements.txt .
This command copies the requirements.txt file from the local host machine into the container's working directory (/app). This file contains a list of Python dependencies required by the application.

RUN pip install --no-cache-dir -r requirements.txt
This installs the Python dependencies listed in requirements.txt using pip. The --no-cache-dir option prevents pip from storing the downloaded packages in the cache, which helps reduce the final image size.

COPY . .
This copies all files and directories from the current project directory on the host machine into the container's working directory. This includes the main application code, such as app.py.

EXPOSE 5000
This informs Docker that the container will listen on port 5000 at runtime. While this does not publish the port to the host machine by itself, it serves as documentation and allows Docker-related tools to recognize the exposed port.

CMD ["python", "app.py"]
This defines the default command to run when the container starts. It launches the Python application by executing app.py using the Python interpreter.

Dockerfile (frontend)

```
FROM nginx:alpine


# Copy static files to nginx html directory

COPY . /usr/share/nginx/html/


# Copy custom nginx configuration

COPY nginx.conf /etc/nginx/conf.d/default.conf


# Expose port 80

EXPOSE 80


# Start nginx

CMD ["nginx", "-g", "daemon off;"]
```

FROM nginx:alpine
This line sets the base image to the official Nginx image based on Alpine Linux, which is known for its small size and efficiency. It includes a pre-installed Nginx server, making it ideal for serving static content.

COPY . /usr/share/nginx/html/
This command copies all files from the current project directory on the host machine into the Nginx web root directory inside the container. These are typically the static files generated by a frontend build process (such as HTML, CSS, JavaScript, and images).

COPY nginx.conf /etc/nginx/conf.d/default.conf
This replaces the default Nginx configuration with a custom one (nginx.conf). This configuration controls important aspects such as:

- SPA (Single Page Application) routing using try_files
- Gzip compression for better performance
- Caching rules for static assets

- Security headers to enhance browser protection

EXPOSE 80
This declares that the container will listen on port 80, the standard HTTP port. This is necessary for serving the frontend to users via a web browser. Note that this does not publish the port; it is typically used in conjunction with docker run -p or in Docker Compose.

CMD ["nginx", "-g", "daemon off;"]
This defines the default command to run when the container starts. It tells Nginx to run in the foreground (daemon off) so that it keeps the container alive and actively serving requests.

Nginx.conf (frontend)

```
server {
    listen 80;
    server_name localhost;

    root /usr/share/nginx/html;
    index index.html;

    # Enable gzip compression
    gzip on;
    gzip_types text/plain text/css application/json application/javascript
text/xml application/xml application/xml+rss text/javascript;

    # Handle all routes
    location / {
        try_files $uri $uri/ /index.html;
    }

    # Cache static assets
    location ~* \.(js|css|png|jpg|jpeg|gif|ico|svg)$ {
        expires off;
        add_header Cache-Control "no-store, no-cache, must-revalidate";
    }

    # Security headers
    add_header X-Frame-Options "SAMEORIGIN" always;
    add_header X-Content-Type-Options "nosniff" always;
    add_header X-XSS-Protection "1; mode=block" always;
}
```

server { ... }

Defines a server block that handles incoming HTTP requests.

listen 80;

Instructs Nginx to listen for HTTP requests on **port 80**, the standard port for web traffic.

server_name localhost;

Specifies the server name. In this case, localhost is used, which is appropriate for local development or default configurations.

root /usr/share/nginx/html;

Sets the **root directory** where Nginx will look for files to serve. This corresponds to the directory where the static frontend files are copied in the Dockerfile.

index index.html;

Specifies the default file (index page) to serve when the root URL (/) is requested.

gzip on;

Enables **Gzip compression**, which reduces the size of transmitted files and improves page load speed for users.

gzip_types ...

Lists the MIME types that should be compressed using Gzip, including common types like CSS, JavaScript, and XML.

location / { try_files $uri $uri/ /index.html; }

Handles routing for **Single Page Applications (SPA)**, such as those built with React or Vue:

- try_files $uri $uri/ /index.html; tells Nginx to:

    1. Serve the requested file if it exists.

    2. Otherwise, fallback to index.html, allowing the frontend framework's router to handle the path.

This is essential for deep linking and client-side routing.

location ~* \.(js|css|png|jpg|jpeg|gif|ico|svg)$ { ... }

This block matches requests for **static assets** like JavaScript, CSS, and image files:

- expires off; disables caching via expiration headers.

- add_header Cache-Control "no-store, no-cache, must-revalidate"; ensures that browsers do not cache these files, which is useful during development or when you want immediate updates to be reflected.

For production, you might change this to enable proper long-term caching and use versioned filenames.

Security Headers

The following headers improve **browser-based security**:

- add_header X-Frame-Options "SAMEORIGIN" always;
  Prevents the site from being embedded in an iframe on another domain, protecting against **clickjacking** attacks.

- add_header X-Content-Type-Options "nosniff" always;
  Prevents browsers from trying to guess ("sniff") the content type, which can help prevent **MIME-type attacks**.

- add_header X-XSS-Protection "1; mode=block" always;
  Enables built-in browser protection against **cross-site scripting (XSS)**.

Docker-compose.yml

```yaml
version: '3.8'

services:
  backend:
    build:
      context: ./backend
      dockerfile: Dockerfile
    container_name: multiple-chat-backend
    ports:
      - "5000:5000"
    environment:
      - FLASK_ENV=production
      - FLASK_DEBUG=0
      - PYTHONUNBUFFERED=1
    restart: unless-stopped
    networks:
      - chat-network
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:5000/health"]
      interval: 30s
      timeout: 10s
      retries: 3

  frontend:
    build:
      context: ./frontend
      dockerfile: Dockerfile
    container_name: multiple-chat-frontend
    ports:
      - "3000:80"
    depends_on:
      - backend
    restart: unless-stopped
    networks:
      - chat-network

networks:
  chat-network:
```

```
    driver: bridge

volumes:
  chat-data:
```

Version

Specifies the version of the Docker Compose file format. Version 3.8 is compatible with Docker 19.03 and newer and is commonly used for both development and production environments.

Services

backend

This service runs the Python-based backend (e.g., Flask API).

- Builds the image using the Dockerfile located in the ./backend directory.
- Sets a custom name for the container to make it easier to reference or debug.
- Maps port 5000 on the host to port 5000 in the container, exposing the backend service.
- Sets environment variables for Flask:
  FLASK_ENV=production: Runs Flask in production mode.
  FLASK_DEBUG=0: Disables debug mode.
  PYTHONUNBUFFERED=1: Ensures logs are output in real time (not buffered).
- Automatically restarts the container unless it is explicitly stopped.
- Connects the service to a custom Docker network for internal communication.
- Performs a health check every 30 seconds by sending a curl request to the backend's /health endpoint.
- The service is considered healthy if the response is successful within 10 seconds, and retries up to 3 times.

frontend

This service runs the static frontend (e.g., React or Vue app served via Nginx).

- Builds the image using the Dockerfile located in the ./frontend directory.
- Sets a custom name for the frontend container.
- Maps port 3000 on the host to port 80 in the container (Nginx listens on port 80).
- Ensures that the frontend service starts **after** the backend service has been started. (Note: this does **not** wait for the backend to become *healthy*.)
- Automatically restarts the container unless it is stopped manually.
- Connects the frontend to the same custom network as the backend for internal communication.

Networks

- Defines a custom bridge network (chat-network) that enables communication between the frontend and backend containers using internal DNS (e.g., the backend can be accessed from the frontend via http://backend:5000).

Volumes

- Declares a named volume (chat-data) for persistent data storage.
- Although not used explicitly in this file, it can be attached to one of the services (e.g., database) in future extensions.

How to Deploy App using Docker

1. Download the source code from Github: https://github.com/ernestteo14/livechat-docker
2. Open your terminal, go to the project directory.
3. Run ‘docker-compose up -d’.
4. Open localhost:3000 in your browser, and boom you already can run the application.
5. To stop, you can run ‘docker-compose down’.
6. For more information, you can read README.md on the github.