Concurrency Strategy

Course Name: Basic Computer Science Practice

Professor: Li Yugang

Class Day/ Time: Monday - Friday / 08.00 - 11.30, 14.10-16.40

Leader: 1820222041 – 温富勝 Alexander Darryl Kristiawan

Member: 1820222021 – 郑国強 Darren Tejaatmaja

1820222030 – 林哲豪 Wilbert Jaya Sucipto

1820222040 – 冯明想 Jesslyn Clarissa Hermanto

Project Topic: Instant Messaging

Designing a concurrency strategy for internet messaging involves ensuring that multiple operations can occur simultaneously without causing data inconsistencies, race conditions, or performance bottlenecks. The general strategy for manage concurrency in an internet messaging system:

1. Threading Model

- **Asynchronous I/O**: Use asynchronous I/O operations to handle multiple incoming and outgoing messages without blocking threads. This is especially useful for handling high volumes of connections simultaneously.
- **Event Loop**: Implement an event loop to manage the non-blocking execution of tasks.
- **Worker Threads**: For CPU-bound tasks (e.g., message encryption or complex data processing), offload work to a pool of worker threads to avoid blocking the main event loop.

2. Concurrency Control

- **Optimistic Concurrency Control**: Use versioning or timestamps to detect and resolve conflicts when multiple operations try to modify the same resource (e.g., a message or user state). If a conflict is detected, retry the operation with updated data.

- **Pessimistic Concurrency Control**: For critical operations (e.g., delivering messages in the correct order), use locks or semaphores to ensure that only one operation can modify the resource at a time. However, use these sparingly to avoid performance degradation.

3. Message Queueing

- **Message Queues**: Implement message queues to decouple message sending and processing. This ensures that messages are processed in order and allows for retries in case of failures.
- **Prioritization**: Implement priority queues for messages that need to be delivered urgently. Lower-priority messages can be processed after the high-priority ones.

4. State Management

- **Distributed State**: For scalability, store state (e.g., user sessions, message metadata) in a distributed data store. Ensure that the data store supports eventual consistency and can handle concurrent read/write operations.
- **Session Consistency**: Use sticky sessions or consistent hashing to ensure that all requests related to a specific user session are handled by the same server, reducing the need for cross-server communication.

5. Data Consistency and Synchronization

- **Atomic Operations**: Ensure that critical operations (e.g., message delivery, status updates) are atomic. Use database transactions or distributed locks to guarantee that these operations are completed successfully or rolled back.
- **Eventual Consistency**: For non-critical operations (e.g., updating user last seen status), use eventual consistency to allow for temporary discrepancies between replicas, improving overall system performance.

6. Load Balancing

- **Horizontal Scaling**: Use load balancers to distribute incoming traffic across multiple servers, preventing any single server from becoming a bottleneck.
- **Shard Data**: Partition data (e.g., by user ID) across different servers or databases to reduce contention and improve access times.

7. Fault Tolerance and Recovery

- **Replication**: Implement data replication across multiple servers or data centers to ensure that messages and state can be recovered in case of a failure.

- **Graceful Degradation**: Design the system to degrade gracefully in case of partial failures, ensuring that essential functions (e.g., messaging) remain operational even if some services are unavailable.
- **Retry Mechanisms**: Implement retry mechanisms with exponential backoff for operations that fail due to temporary issues (e.g., network latency or server overload).

8. Testing and Monitoring

- **Concurrency Testing**: Perform stress testing and concurrency testing to identify bottlenecks, race conditions, and deadlocks in the system.
- **Monitoring Tools**: Use monitoring tools to track system performance, detect anomalies, and trigger alerts in case of issues.

This strategy aims to balance performance, consistency, and scalability, ensuring that the messaging system can handle high concurrency while maintaining reliability and data integrity.

Testing Strategy

1. Functional Testing

Basic Messaging:

- Start the server.
- Launch two clients.
- Test sending a message from Client A to Client B.
- Verify that Client B receives the message correctly.

Connection Handling:

- Ensure that clients can connect to the server without errors.
- Test disconnecting and reconnecting clients to see if connections are managed properly.

2. Error Handling

Message Loss:

Simulate network interruptions (e.g., disconnect one client while sending a message) and check how the app handles the situation. Ensure that appropriate error messages are shown if needed.

Invalid Messages:

Try sending corrupted or malformed messages and see if the app can handle these gracefully without crashing.

3. Performance Testing

Basic Load:

Test how the app performs with multiple messages sent in quick succession. Ensure that messages are not lost and are displayed in the correct order.

4. User Interface (UI) Testing

Basic Functionality:

Verify that all user interface elements (buttons, text fields) are functional and respond to user interactions as expected.

Message Display:

Ensure that messages are displayed correctly in the chat window and that the UI updates in real-time.

5. Usability Testing

Ease of Use:

- Check if users can easily send and receive messages.
- Make sure the interface is intuitive and that users can easily navigate through the app.

6. Compatibility Testing

Basic System Compatibility:

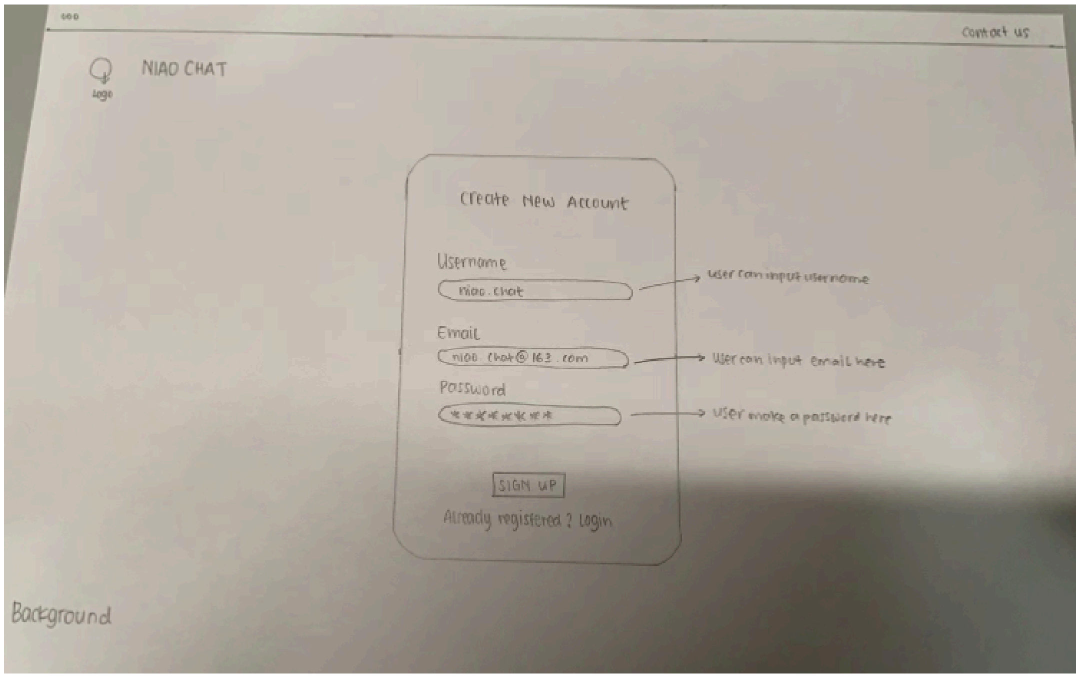
- Test the application on different machines or operating systems (if possible) to ensure it runs smoothly across environments.

7. **Crash Testing**

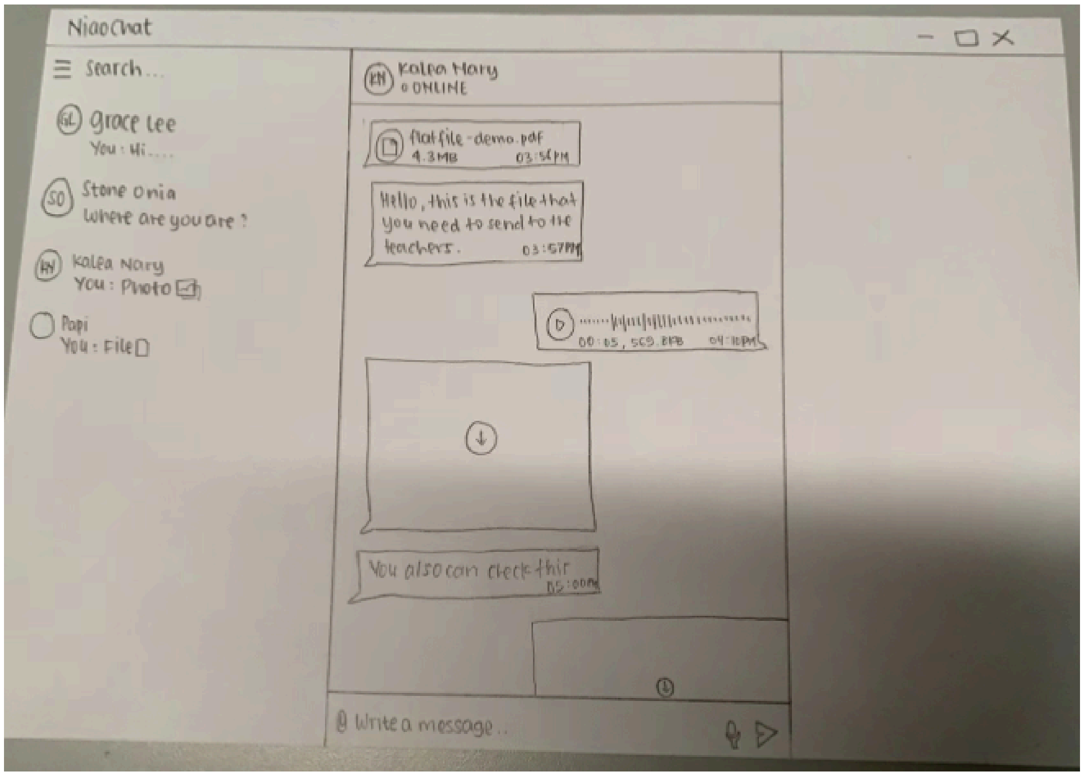
Unexpected Behavior:

Forcefully close the application or server during operation and check if the app handles unexpected terminations gracefully and can be restarted without issues.

UI Sketches



Contact us
logo
NIAO CHAT
Create New Account
Username
niao.chat
user can input username
Email
niao.chat@163.com
User can input email here
Password
****
User make a password here
SIGN UP
Already registered ? Login
Background



NiaoChat
Search...
GL Grace Lee
You : Hi....
SO Stone Onia
Where are you are ?
KN Kalea Nary
You : Photo
Papi
You : File
KN Kalea Nary
o ONLINE
flat file-demo.pdf
4.3MB 03:56PM
Hello, this is the file that you need to send to the teachers. 03:57PM
00:05, 509.8KB 04:10PM
You also can check this 05:00PM
Write a message...

Layout Overview:

- **Main Window:** The app's main window is divided into two main sections: the Contacts List on the left and the Chat Window on the right.

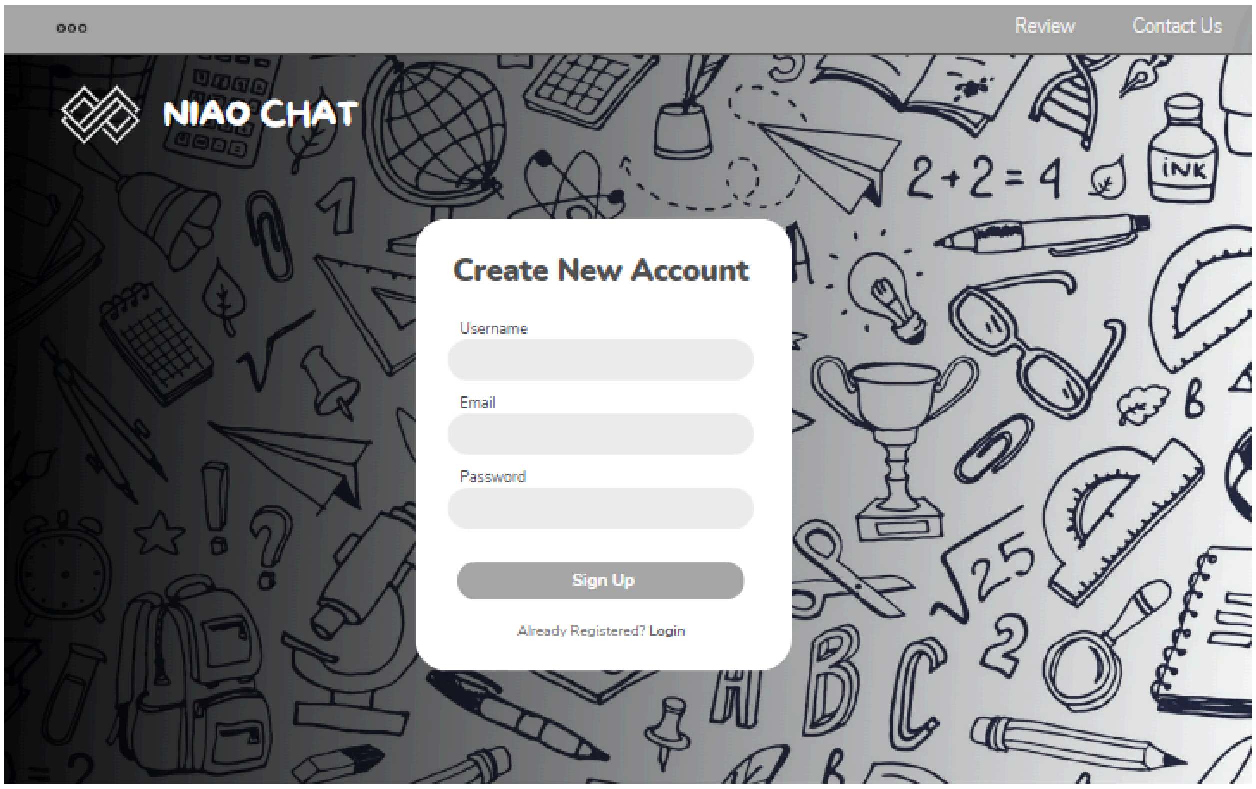
Contacts List (Left Panel):

- **Header:** At the top of the contacts list, there's a search bar where users can search for contacts by name.
- **Contact Items:** Below the search bar, a vertical list displays the user's contacts.

Chat Window (Right Panel):

- **Header:** The top of the chat window displays the name of the contact you are chatting with, along with their online status.
- **Chat :** The main area of the chat window is where all messages are displayed in a scrolling format.
- **Message Input Field:** At the bottom of the chat window, there's a text input field where users can type their messages.
    - **Send Button:** A simple "Send" button is located to the right of the input field.
    - **Attach Button:** Next to the send button, there's a small paperclip icon that allows users to attach files or images.

Design plan:


ooo
Review
Contact Us
NIAO CHAT
Create New Account
Username
Email
Password
Sign Up
Already Registered? Login