**1. Conceptual Questions**

**What is the purpose of streams in MultiChain, and how are they used in your project?**

Streams in MultiChain are used to store and organize data. In our project:

* weather\_alerts stream stores weather updates.
* user\_registration stream stores user credentials and roles.

**Why did you choose MultiChain for this project instead of another blockchain platform?**

MultiChain is efficient for private blockchains, supports data streams, and allows fine-grained access control, which fits our project's needs.

**Explain the difference between publish and subscribe in MultiChain.**

* Publish: Adds data to a stream (e.g., weather alerts).
* Subscribe: Allows a node to access and monitor a stream's data.

**What problem does your project solve, and how is blockchain technology integral to its solution?**

Our project provides secure weather alerts and user management. Blockchain ensures data immutability, transparency, and secure access.

**How does your project ensure secure access and authorization?**

Users are authenticated by matching credentials in the blockchain. Roles (e.g., admin, viewer) control access permissions.

**2. Implementation-Specific Questions**

**How does the authentication process work in your project?**

User data (username, password, role) is stored in user\_registration stream. During login, credentials are matched with the stream data.

**Why is hashing the password important, and why hasn’t it been implemented in your code?**

Hashing secures passwords against theft. It hasn’t been implemented yet but should be added for better security.

**Explain how weather data is fetched and published on the blockchain.**

Weather data is fetched from OpenWeather API. If severe conditions (e.g., Rain) are detected, alerts are published to the weather\_alerts stream.

**What challenges arise from running services on different operating systems?**

Network connectivity and compatibility issues may occur. These are handled by proper configuration and consistent API usage.

**3. Blockchain-Specific Technical Questions**

**What are the advantages of using MultiChain streams over traditional databases?**

* Immutable records ensure data integrity.
* Decentralized storage prevents single points of failure.
* Transparent access for authorized nodes.

**How does your project achieve immutability and transparency?**

Data written to the blockchain cannot be altered, and all nodes with access can verify the stored data.

**How would you integrate smart contracts to enhance functionality?**

Smart contracts could automate tasks like role-based access or sending alerts only to specific users.

**What potential vulnerabilities exist, and how can they be mitigated?**

* Passwords are stored in plaintext. **Mitigation**: Use hashing.
* Unauthorized node access. **Mitigation**: Use strong node authentication.

**4. Critical Thinking Questions**

**If a user’s credentials are compromised, how can the blockchain be updated to prevent access?**

Revoke the compromised account's role and update the blockchain with new credentials.

**What would happen if a node fails in the MultiChain network?**

Other nodes can still access the blockchain since the data is replicated.

**How would you scale the project for more users and alerts?**

Increase the number of nodes, optimize data storage, and batch process alerts to handle higher loads.

**What are the limitations of your project, and how can they be improved?**

* No password hashing: Add it.
* Limited roles: Expand with more granular roles.
* No real-time alerting: Implement real-time notifications.

**5. Practical Coding and Debugging Questions**

**Write code to hash and store the user password securely.**

python

Copy code

import hashlib

def hash\_password(password):

return hashlib.sha256(password.encode()).hexdigest()

hashed\_password = hash\_password("password123")

print(hashed\_password) # Store this instead of plaintext

**Modify the project to allow password updates.**

1. Authenticate the user.
2. Hash the new password.
3. Publish updated credentials to the user\_registration stream.

**Debugging Scenario: No weather alerts are published.**

1. Check API response for weather data.
2. Verify if severe weather conditions are correctly detected.
3. Ensure mc.publish to the blockchain is successful.

**6. Comparison Questions**

**Compare blockchain to a traditional database in your project.**

* **Blockchain**: Immutable, decentralized, transparent.
* **Database**: Mutable, centralized, faster for basic queries.

**How would the project differ if implemented on Ethereum?**

* Use smart contracts for logic.
* Higher costs due to gas fees.
* Public blockchain would reduce privacy.

**7. Exam-Specific Scenario Questions**

**How would you handle tampered stream data?**

Check blockchain hashes to detect tampering. Implement role-based access control to prevent unauthorized publishing.

**How would you implement role-based alerts?**

Store roles in the user\_registration stream. Filter users based on roles when sending alerts.

**8. Advanced Questions**

**How would you integrate Merkle Trees for verifying alerts?**

* Store the root hash of all alerts in the blockchain.
* Use Merkle proofs to verify individual alerts without storing the entire dataset.