**Basic Input Validation Questions**

1. What validation will you add to ensure username and password are not empty when registering?
2. How can you enforce a minimum password length or complexity (for example, at least one number and one special character)?
3. What should happen if a user tries to deposit or withdraw a negative amount?
4. What validation can prevent a user from depositing or withdrawing a zero amount?
5. What error message should be returned if a client sends an incomplete JSON body (for example, missing amount or account\_id)?
6. How can you validate that the username field does not contain spaces or special characters?
7. How will you validate if the amount entered is a valid number and not a string like "five thousand"?
8. What should the API return if a user provides an invalid or malformed JSON request body?
9. How can you check that the account ID passed in /deposit or /withdraw actually belongs to the logged-in user?
10. How can you prevent users from entering extremely large amounts (e.g., ₹10 crore) that might crash the system?

**Database-Level Validation Questions**

1. How does the system handle registering an already existing username? Can you improve this using unique constraint validation in SQLAlchemy?
2. What happens if the amount in /deposit is sent as a string instead of a float? How can you handle that gracefully?
3. How can you enforce that the balance column never becomes negative in the database, not just in the Python logic?
4. How will you ensure the relationship between User → Account → Transaction remains consistent even after multiple transactions?
5. What would happen if a transaction is recorded without an associated account? How can you prevent that?
6. How can you validate that when a user is deleted, all related accounts and transactions are also cleaned up properly?

**Business Logic Validation Questions**

1. What validation will you perform before allowing a withdrawal (for example, checking for sufficient balance)?
2. How will you prevent one user from accessing another user’s account or transaction history?
3. Should a user be allowed to create multiple bank accounts? If not, how would you enforce that in code?
4. Suppose there is a limit of ₹10,000 per withdrawal — how will you validate this before updating the balance?
5. How will you handle cases where users perform multiple simultaneous deposits or withdrawals (race conditions)?
6. How can you ensure that each transaction correctly updates the balance even if the app crashes midway?
7. What happens if a user tries to withdraw from a non-existent account? What validation should be added?
8. How will you handle rounding or floating-point precision errors (for example, ₹0.009999)?
9. How can you validate that the logged-in user only accesses their own data through the token identity?
10. How can you validate that a user cannot perform a withdrawal on an account they don’t own?

**JWT and Authentication Validation Questions**

1. How can you ensure that /deposit, /withdraw, /balance, and /transactions cannot be accessed without a JWT token?
2. What response should your API send if a user provides an expired JWT token?
3. How will your API respond if a user tries to access a protected endpoint with a tampered (invalid) token?
4. How does get\_jwt\_identity() help validate which user is performing the operation?
5. How can you make sure tokens are unique per user session or have expiration times for better security?
6. What should happen if a user logs out — how can you invalidate their token?
7. How can you detect and handle if a token was stolen or reused by another device?
8. How can you ensure sensitive routes are accessible only after verifying both authentication (JWT) and authorization (user role)?
9. What is the difference between authentication and authorization in the context of JWT validation?

**Error Handling and Response Validation**

1. Why is it important to return consistent JSON responses for validation errors (for example, { "error": "Invalid amount" })?
2. Which HTTP status codes would you use for:
   * Missing fields
   * Unauthorized access
   * Insufficient balance
   * Successful deposit or withdrawal
3. How can you design a global error handler in Flask to manage all validation errors?
4. How can you ensure that the API never crashes even if unexpected input is received (for example, missing keys)?
5. How would you implement a function to validate request fields before performing any business logic?