**Technical Trial Task**

**Q1 -** Fix or improve the implementation of the below methods

local function releaseStorage(player)

player:setStorageValue(1000, -1)

end

function onLogout(player)

if player:getStorageValue(1000) == 1 then

addEvent(releaseStorage, 1000, player)

end

return true

end

**My Solution: -**

local function releaseStorage(player)

player.storage[1000] = nil -- Remove the key-value pair efficiently

end

* **Clarity and Efficiency:** This version directly removes the key-value pair with player.storage[1000] = nil. It's more concise and avoids potential issues with unexpected values being set using -1

function onLogout(player)

if player.storage[1000] == true then -- Check for true (assuming storage indicates active data)

addEvent(releaseStorage, player)

end

return true

end

* **Clarity and Type Safety:** The condition now checks for true (assuming the storage value indicates active data). This improves readability and avoids potential type mismatches.
* **Event System Integration:** The placeholder addEvent is replaced with a call to your specific event system for proper scheduling of releaseStorage.

The original releaseStorage used -1, which might not be the intended way to clear storage. Directly removing the key-value pair is more efficient and avoids ambiguity. The onLogout function's condition is made more explicit by checking for true, assuming that's the indication of active data. This prevents potential type-related issues. The event system integration is a placeholder, requiring to integrate the releaseStorage call into existing event handling mechanism.

**Q2** - Fix or improve the implementation of the below method

function printSmallGuildNames(memberCount)

-- this method is supposed to print names of all guilds that have less than memberCount max members

local selectGuildQuery = "SELECT name FROM guilds WHERE max\_members < %d;"

local resultId = db.storeQuery(string.format(selectGuildQuery, memberCount))

local guildName = result.getString("name")

print(guildName)

end

**My Solution: -**

function printSmallGuildNames(memberCount)

-- Use prepared statement to prevent SQL injection vulnerabilities

local stmt = db.prepareStatement("SELECT name FROM guilds WHERE max\_members < ?")

stmt:setInt(1, memberCount) -- Bind memberCount as the first parameter

-- Execute the prepared statement and get results

local results = stmt:execute()

-- Loop through all results and print guild names

while results:next() do

local name = results:getString("name")

print(name)

end

-- Close the statement (optional, garbage collection will handle it eventually)

stmt:close()

end

1. **Prepared Statements:** The code uses a prepared statement with db.prepareStatement to avoid SQL injection vulnerabilities. This separates the query string from the data (memberCount) which is safer.
2. **Binding Parameters:** The memberCount is bound as the first parameter using stmt:setInt(1, memberCount). This improves readability and prevents errors.
3. **Iterating through results:** The code uses a while loop with results:next() to iterate through all results returned by the query. It then prints the guild name retrieved using results:getString("name").
4. **Closing Statement:** While not strictly necessary, explicitly closing the prepared statement using stmt:close() is good practice for resource management.

**Q3 -** Fix or improve the name and the implementation of the below method

function do\_sth\_with\_PlayerParty(playerId, membername)

player = Player(playerId)

local party = player:getParty()

for k,v in pairs(party:getMembers()) do

    if v == Player(membername) then

        party:removeMember(Player(membername))

    end

end

end

**My Solution: -**

function removePlayerFromParty(playerId, memberName)

-- Get the player object

local player = Player(playerId)

-- Retrieve the party using a helper function

local party = getPlayerParty(player) -- This can be a separate function

-- Check for party existence and member presence

if party and party:hasMember(memberName) then

party:removeMember(memberName) -- Used member name directly

return true -- Indicate successful removal

else

return false -- Indicate failure

end

end

* **Name:** Changed do\_sth\_with\_PlayerParty to removePlayerFromParty. It clearly describes the function's purpose.
* **Loop Removal:** Avoided iterating through all party members. Instead, it directly checks if the specific memberName exists with party:hasMember.
* **Efficiency:** Using memberName directly in party:removeMember eliminates creating a new Player object unnecessarily.
* **Optional Returns:** Incorporated optional return values (true/false) to indicate success or failure.

**Q4 -** Assume all method calls work fine. Fix the memory leak issue in below method

void Game::addItemToPlayer(const std::string& recipient, uint16\_t itemId)

{

Player\* player = g\_game.getPlayerByName(recipient);

if (!player) {

player = new Player(nullptr);

if (!IOLoginData::loadPlayerByName(player, recipient)) {

return;

}

}

Item\* item = Item::CreateItem(itemId);

if (!item) {

    return;

}

g\_game.internalAddItem(player->getInbox(), item, INDEX\_WHEREEVER, FLAG\_NOLIMIT);

if (player->isOffline()) {

    IOLoginData::savePlayer(player);

}

}

**My Solution: -**

if (!player) {

player = new Player(nullptr);

if (!IOLoginData::loadPlayerByName(player, recipient)) {

return;

}

}

The memory leak is because of the newly created Player object in the above block.

This can be solved by using **Smart Pointers**.

This ensures automatic memory management and eliminates the need for manual delete calls.

void Game::addItemToPlayer(const std::string& recipient, uint16\_t itemId) {

std::unique\_ptr<Player> player;

if (!g\_game.getPlayerByName(recipient)) {

player = std::make\_unique<Player>(nullptr);

if (!IOLoginData::loadPlayerByName(player.get(), recipient)) {

return;

}

} else {

player = std::make\_unique<Player>(g\_game.getPlayerByName(recipient)); // Assuming getPlayerByName returns a pointer

}