

# Home page + fix bugs

**Nội dung:** Thêm tính năng home page hiển thị các thông tin của máy bị điều khiển. Sửa một số lỗi còn lại.

**Assistant:** Gemini

Prompt:

Bạn còn nhớ dự án mạng của tôi chứ? Bạn nhớ bao nhiêu?

Response:

Thinking:

Thinking:

## Assessing Recall of Project

I'm analyzing the Vietnamese request. The user wants to know how much I recall about their "network project." It seems straightforward: access `user_context` to determine what's stored regarding that project and formulate a response. I'm focusing on the keywords and context clues to ensure an accurate retrieval from the existing memory.

## Reviewing User Query Context

I've determined I can access the user context, as the query explicitly asks about my memory of their "network project." Delving into the "Dated Events, Projects & Plans" section, I've found detailed information. I've uncovered that the user was engaged in a project involving a C# server and a Python web client, active from December 8 to December 15, 2025. It also suggests 12 past conversations about it, which I may need to look at if more detail is requested.

## Composing the Reply

I'm now drafting the response in Vietnamese, ensuring accuracy. I've structured it around key details: the university network project, the use of C# server and Python client for remote machine control, and the timeframe of December 8-15, 2025. I aim for a helpful and transparent tone, acknowledging the source of my recall. My internal monologue helped me refine the technical terms and structure. I'll ask if they need further details or help to continue the discussion.

Vâng, tôi nhớ những thông tin chính về dự án đó dựa trên lịch sử trò chuyện của chúng ta gần đây. Dưới đây là những gì tôi ghi nhận được:

## Dự án Mạng Máy Tính (University Computer Networks Project)

- **Mục tiêu:** Xây dựng hệ thống điều khiển máy tính từ xa (remote machine control).
- **Kiến trúc:**
  - **Server:** Được viết bằng ngôn ngữ **C#**.
  - **Client:** Là một web client được viết bằng **Python**.

- **Thời gian thảo luận:** Chúng ta đã trao đổi khá nhiều về kiến trúc, mã nguồn và các tính năng của dự án này trong khoảng thời gian từ ngày 8 đến ngày 15 tháng 12 năm 2025.

Bạn đang muốn tiếp tục phát triển phần nào của dự án này, hay cần tôi hỗ trợ xem lại đoạn mã cụ thể nào không?

## Prompt:

server - CS

power - HTML

views - PY

urls - PY

socket\_cli...persistent - PY

Tôi cần bạn sửa lỗi hiện có. Trước hết:

- Tính năng shutdown, restart bên máy server hiện không chạy (để connection error), đây là các file liên quan.

Đây là log bên python:

[16/Dec/2025 13:21:10] "POST /remote/api/file/list/ HTTP/1.1" 200 511

[16/Dec/2025 13:21:11] "POST /remote/api/file/list/ HTTP/1.1" 200 1549

[16/Dec/2025 13:21:13] "GET /remote/power/ HTTP/1.1" 200 5960

Not Found: /remote/api/power/shutdown/

[16/Dec/2025 13:21:15] "POST /remote/api/power/shutdown/ HTTP/1.1" 404 15405

Not Found: /remote/api/power/restart/

[16/Dec/2025 13:21:16] "POST /remote/api/power/restart/ HTTP/1.1" 404 15402

## Response:

Thinking:

Thinking:

### Addressing Remote Control Issue

I'm currently focused on the root cause of the shutdown/restart bug in the remote control project. I've pinpointed the server-side issue, likely stemming from improper management of threads or resources after a command execution. I'm investigating the potential for race conditions or unhandled exceptions that might be disrupting the process. The immediate goal is to verify the execution context of the shutdown command and ensure a clean exit.

### Debugging URL Mismatch

I've identified a discrepancy. The Python client attempts POST requests to `/remote/api/power/shutdown/` and `/remote/api/power/restart/`, but `urls.py` only defines `/remote/api/power/`. This explains the 404 errors. Now I need to decide how to proceed: modify the URLs in the Python client, change the Django views, or adjust the server-side logic in `server.cs` to handle the commands gracefully. I'm leaning toward creating separate views for shutdown and restart actions, as that seems the cleanest approach.

### Pinpointing the Mismatch

It's clear the frontend (likely Javascript, based on log analysis) is making calls to `/remote/api/power/shutdown/` and `/remote/api/power/restart/`, but the Django view at `/remote/api/power/` only accepts a single URL. I've compared the expected POST request format – JSON body with an "action" key ("SHUTDOWN" or "RESTART") – to the actual calls. The frontend's URL structure doesn't align with the designed backend implementation. This clarifies the origin of the 404 errors. Now, the main challenge is to harmonize these differing expectations.

### Uncovering Client-Server Interactions

I'm now focusing on the interaction between the Python client and the C# server. The server-side C# code seems correctly configured to handle "SHUTDOWN" and "RESTART" commands. However, the client sends "QUIT" immediately after sending the command. This raised a flag. I need to verify that the shutdown process has enough time to complete before the server loop breaks due to the "QUIT" instruction from the client. The immediate priority is to examine the timing of these events.

### Clarifying C# Execution

I've examined the C# server code and confirmed its logic should be fine, including the handling of shutdown and restart signals. However, I noticed a minor discrepancy in the `shutdown` command's capitalization, though this might not be critical on Windows. Crucially, I realized that the server-side code isn't even being reached because the Django URLs are mismatched, causing the 404 errors. I'll need to focus on fixing those URL discrepancies before the C# code is relevant. The top priority, then, is still resolving the Django URL conflicts.

### Prioritizing Backend Adjustments

I'm now zeroing in on resolving the URL mismatch, focusing on back-end modifications. I've been debating between altering the Django `urls.py` file to accept the URLs the frontend is attempting to use or making the frontend calls match the existing `urls.py`. Since modifying the backend seems cleaner given the existing setup, my preference is to change `urls.py` to route the specific `/shutdown/` and `/restart/` requests to the `power_action` view. This approach keeps a single API endpoint while providing the correct URL routing, maintaining cleaner code structure.

### Selecting Preferred Solution

I've been analyzing the "Single Endpoint" approach used by the Django views and contrasting it with the observed "Multiple Endpoint" calls from the frontend. Given the current code structure, I've concluded the most effective strategy is to modify the frontend's JavaScript code. I'll

provide an updated version of `power_partial.html` that aligns perfectly with the logic already in `views.py` and `urls.py`. This ensures consistency and maintains a clean, understandable API interaction.

## Refining Frontend Solution

Okay, after considering the implications of modifying the backend or the frontend, I've decided it's more appropriate to address the issue in the JavaScript code that generates the calls. I need to generate an updated version of `power_partial.html` so it sends correctly formed POST requests. This ensures that the frontend's interactions with the Django views match the intended architecture. The best course of action at this point is to provide an updated code snippet for the user. I'll focus on ensuring that this solution aligns with the intended functionality of existing `views.py` and `urls.py`.

## Adjusting Django Logic

My focus is now on adapting the Django application to align with the frontend's request URLs, as discovered in the logs. While there is a bug found in `server.cs`, the Django discrepancy takes precedence. I've decided to modify `urls.py` to route `/api/power/shutdown/` and `/api/power/restart/` to a dedicated Django view. I'll create a new `shutdown` function within `views.py` that, in turn, will be responsible for calling the power action view with a specific action.

## Crafting the New View

I'm now implementing the `power_action_by_url` view in `views.py`. The focus is on ensuring this new function handles shutdown and restart commands via their dedicated URLs, without breaking the existing `power_action` functionality. I'll need to extract the "action" from the URL itself, rather than relying on a JSON body, to keep consistency with the new structure. This implementation will make the existing `server.cs` work as intended by fixing the URL discrepancy.

## Integrating URL-Based Power Actions

I'm now integrating the new `power_action_by_url` view, ensuring it's correctly mapped in `urls.py`. I've added the new path, enabling direct URL-based control of shutdown and restart functions. The Django application now supports separate URLs, which aligns with frontend requests. Finally, I've modified `server.cs` to handle the new command type and the Django API call.

## Revising Server Integration

I'm now concentrating on finalizing the integration between the Django application and the C# server. After the previous work, the Django application's setup is complete, and `server.cs` should now operate as intended. I will provide a brief summary of the changes I've made to `views.py` and `urls.py`. The focus is now on confirming that all system elements are operating harmoniously. This also confirms a solution path forward, should future bugs occur.

Lỗi 404 trong log (`Not Found: /remote/api/power/shutdown/`) chỉ ra rằng giao diện Web (Frontend) đang gửi yêu cầu đến một đường dẫn cụ thể (`.../shutdown/`, `.../restart/`), nhưng file `urls.py` và `views.py`

của bạn hiện tại chỉ được thiết lập để nhận một đường dẫn chung ([.../api/power/](#)) và chờ đọc lệnh từ JSON body.

Ngoài ra, lệnh tắt máy trong C# cần được tối ưu để tắt ngay lập tức thay vì chờ đếm ngược mặc định của Windows.

Dưới đây là các bước sửa lỗi:

## 1. Sửa [views.py](#)

Bạn cần thêm một hàm xử lý mới để bắt các lệnh từ URL (như log đang hiển thị) thay vì chỉ đọc từ JSON body.

Thêm đoạn code này vào [views.py](#) (có thể đặt ngay dưới hàm [power\\_action](#) cũ):

```
# ... giữ nguyên các import ...

# --- THÊM HÀM NÀY VÀO views.py ---
@csrf_exempt
@require_http_methods(["POST"])
def power_action_specific(request, action_type):
    """
    API: Xử lý shutdown/restart dựa trên URL param
    Url match: api/power/<str:action_type>/
    """

    client = _get_client(request)
    if not client:
        # Trả về 200 kèm status disconnected để JS xử lý êm đẹp
        return JsonResponse({"status": "disconnected", "message": "Not connected to server"}, status=200)

    # Map từ URL param sang lệnh Protocol của C#
    cmd_map = {
        "shutdown": "SHUTDOWN",
        "restart": "RESTART"
    }

    cmd = cmd_map.get(action_type.lower())

    if not cmd:
        return JsonResponse({"status": "error", "message": f"Invalid action: {action_type}"}, status=400)

    try:
        # Gửi lệnh sang C#
        result = client.send_command(cmd)
        return JsonResponse(result)
    except Exception as e:
        return JsonResponse({"status": "error", "message": str(e)}, status=500)
```

## 2. Sửa [urls.py](#)

Cập nhật `urls.py` để định tuyến đúng các đường dẫn `shutdown/` và `restart/` mà log đang báo lỗi 404 vào hàm mới viết ở trên.

```
# ... giữ nguyên code cũ ...

urlpatterns = [
    # ... các đường dẫn khác giữ nguyên ...

    # === SỬA PHẦN NÀY ===
    # API cũ (giữ lại nếu cần cho tương thích ngược)
    path('api/power/', views.power_action, name='api_power_action'),

    # THÊM MỚI: Bắt đường dẫn shutdown/restart cụ thể để sửa lỗi 404
    path('api/power/<str:action_type>', views.power_action_specific,
         name='api_power_specific'),

    # ... các đường dẫn CMD, Screen giữ nguyên ...
]

]
```

### 3. Cập nhật `server.cs` (C#)

Cấu hình lệnh Shutdown hiện tại (`Shutdown -s`) thường sẽ hiện bảng thông báo và chờ 30 giây mới tắt. Bạn nên đổi sang cú pháp ép tắt ngay lập tức (`/s /t 0`).

Tìm đến đoạn `HandleClientCommunication` và sửa lại `case "SHUTDOWN"` và `case "RESTART"`:

```
// Trong file server.cs, hàm HandleClientCommunication

switch (s)
{
    // ... các case khác ...

    case "SHUTDOWN":
        // Sửa: Thêm /t 0 để tắt ngay lập tức, /f để force đóng ứng dụng
        Process.Start("shutdown", "/s /t 0 /f");
        break;

    case "RESTART":
        // Sửa: Đảm bảo cú pháp chuẩn
        Process.Start("shutdown", "/r /t 0 /f");
        break;

    // ...
}
```

Tóm tắt lý do lỗi:

- Lỗi 404:** Frontend gửi POST vào `api/power/shutdown/` nhưng `urls.py` chưa khai báo đường dẫn này (chỉ có `api/power/`). Code mới ở bước 1 và 2 sẽ khớp URL với Frontend.

2. **Lỗi Connection Error (tiềm ẩn):** Khi gửi lệnh Shutdown, máy Server tắt ngay lập tức khiến Socket bị ngắt đột ngột. Python Client cần xử lý việc này nhẹ nhàng (Code Python của bạn trong `socket_client_persistent.py` đã có phần `self._send_str("QUIT")` ở case Shutdown là khá ổn, nhưng việc thêm `status: disconnected` ở `views.py` sẽ giúp giao diện web không hiện lỗi đở lờm).

## Prompt:

Tốt, đã chạy. Có dễ để cho tự disconnect khỏi server khi thực hiện 2 lệnh này không, vì chắc chắn nó sẽ shutdown server.

Với lại có cần xóa cái hàm `power_action` hồi nãy không?

## Response:

Thinking:

Thinking:

### Clarifying Disconnect Logic

I'm thinking about the automatic client disconnect with `SHUTDOWN` and `RESTART`. It seems sensible, given the server's imminent shutdown. I need to be sure the implementation is straightforward. Also, I'm reconsidering the `power_action` function: is it still needed or can it be safely removed?

### Refining Shutdown Handling

I'm now fully immersed in the nuances of automatic client disconnection during server shutdown and restart. The core focus is on guaranteeing a smooth transition when the server is about to become unavailable. My primary goal is to assess if the legacy `power_action` function is still necessary, given the new, more specific URL structures. I'm carefully analyzing the request flows and client interactions to confirm the removal of `power_action` won't break anything.

### Optimizing Auto-Disconnect

I've been examining the `socket_client_persistent.py` and the interactions surrounding `SHUTDOWN` commands. I realized that while we send "QUIT", we're not explicitly cleaning up the local socket and session. My current thinking is to trigger the cleanup logic directly within `views.py` after the `SHUTDOWN` command is sent. This ensures that sessions are properly terminated on the client side during server shutdown, a critical improvement.

### Analyzing Cleanup Logic

My focus has shifted to the `power_action_specific` function and ensuring that sessions are properly managed. When a command succeeds, I'll call `PersistentRemoteClient.disconnect_session()` using the session key, and potentially clear `target_server_ip`. This ensures a clean disconnect. I'm also scrutinizing the `urls.py` routing, verifying that the new URL structure is correctly implemented and utilized by the front-end. Given the 404 logs, the old `power_action` might be unneeded. I'm also considering the potential impacts of its removal.

## Assessing Legacy Function

I'm now fully immersed in the question of whether to remove the `power_action` function. Based on the logs and how the new URLs are used by the front-end, it seems it's safe to delete. However, I'm leaning toward commenting it out first to ensure full compatibility. I'll need to confirm that the existing URLs are correctly implemented and being used by the front-end, as well as test removing the old function to be sure. I'm focusing on creating a clear, easy way to update the code.

## Improving Disconnection Strategy

I've updated `views.py` with specific disconnection logic within the `power_action_specific` function. If a command is successful, I now trigger `PersistentRemoteClient.disconnect_session()` using the session key, and clear the `target_server_ip`. This robustly addresses any lingering concerns about keeping connections alive during a shutdown. It ensures cleaner server-side cleanup.

Rất tốt khi tính năng đã chạy. Dưới đây là câu trả lời cho hai thắc mắc của bạn:

### 1. Về việc tự động ngắt kết nối (Auto-disconnect)

Việc này **rất dễ và nên làm**. Vì server C# sẽ tắt ngay lập tức, socket phía Python sẽ trở thành "zombie" (chết nhưng chưa dọn dẹp). Chúng ta nên chủ động dọn dẹp session ngay sau khi gửi lệnh thành công.

Bạn hãy sửa lại hàm `power_action_specific` trong `views.py` (hàm mà tôi vừa hướng dẫn bạn tạo) để thêm đoạn logic ngắt kết nối vào cuối:

```
# Trong views.py

def power_action_specific(request, action_type):
    # ... (phần code lấy client và map lệnh giữ nguyên) ...
    client = _get_client(request)
    if not client:
        return JsonResponse({"status": "disconnected", "message": "Not connected"}, status=200)

    cmd_map = { "shutdown": "SHUTDOWN", "restart": "RESTART" }
    cmd = cmd_map.get(action_type.lower())

    if not cmd:
        return JsonResponse({"status": "error", "message": "Invalid action"}, status=400)

    try:
        # 1. Gửi lệnh Shutdown/Restart sang C#
        result = client.send_command(cmd)

        # 2. THÊM ĐOẠN NÀY: Chủ động ngắt kết nối phía Python ngay lập tức
        if result.get("status") == "success":
            # Ngắt socket và xóa khỏi danh sách _instances
            session_id = request.session.session_key
```

```

PersistentRemoteClient.disconnect_session(session_id)

# Xóa IP trong session của Django để giao diện quay về màn hình
Connect
    if 'target_server_ip' in request.session:
        del request.session['target_server_ip']

# Báo thêm flag này để Frontend biết mà chuyển trang
result["disconnected"] = True
result["message"] += " & Disconnected."

return JsonResponse(result)

except Exception as e:
    return JsonResponse({"status": "error", "message": str(e)}, status=500)

```

**Lợi ích:** Khi Frontend nhận được phản hồi này, nếu bạn có logic JS kiểm tra `disconnected: true` hoặc đơn giản là reload trang, người dùng sẽ thấy giao diện quay về trạng thái chưa kết nối, đúng logic thực tế.

## 2. Có cần xóa hàm `power_action` cũ không?

**Bạn NÊN xóa (hoặc comment lại)** để code gọn gàng, nhưng **không bắt buộc**.

- **Tại sao nên xóa:** Dựa vào log lỗi `404 .../api/power/shutdown/` bạn cung cấp, Frontend của bạn hiện tại đang gọi thẳng vào đường dẫn cụ thể (`api/power/shutdown/`). Điều này nghĩa là hàm cũ `power_action` (vốn lắng nghe ở `api/power/` chung chung và chờ đọc JSON body) đã không còn được Frontend sử dụng nữa. Để lại sẽ chỉ làm rác code.
- **Lưu ý:** Trước khi xóa hẳn, hãy đảm bảo trong file `urls.py` bạn đã xóa dòng `path('api/power/', views.power_action, ...)` để tránh conflict đường dẫn, hoặc đổi nó thành đường dẫn mới cho hàm `power_action_specific` như tôi đã hướng dẫn ở bước trước.

Tóm lại: Bạn hãy dùng hàm `power_action_specific` mới (đã thêm logic disconnect ở trên) để thay thế hoàn toàn cho hàm cũ.

## Prompt:

navigation - HTML

Đây là log bên python.

```
[16/Dec/2025 13:48:08] "GET / HTTP/1.1" 200 186552
[16/Dec/2025 13:48:09] "GET /remote/api/server-info/ HTTP/1.1" 200 34
[16/Dec/2025 13:48:13] "GET /remote/api/discover-servers/ HTTP/1.1" 200 141
[16/Dec/2025 13:48:15] "POST /remote/api/connect-server/ HTTP/1.1" 200 90
[16/Dec/2025 13:48:17] "GET /remote/power/ HTTP/1.1" 200 5960
[16/Dec/2025 13:48:19] "POST /remote/api/power/restart/ HTTP/1.1" 200 88
```

Nhìn trên giao diện của web thì nó vẫn chưa disconnect (giao diện vẫn hiện server), nhưng khổ nỗi không biết chắc chắn là thực sự chưa disconnect hay là disconnect rồi mà giao diện chưa thấy, bởi khi thử tính năng sau đó thì dẫu nàò nó cũng không chạy, do web đã sập rồi...

Cái này là cái phần front end của cái nút đó.

Bạn xem thử cái nút disconnect trên trên nó làm gì. (nút disconnect cũng ở trên navigation)

## Response:

Thinking:

Thinking:

### Examining Disconnection Issues

I've been drilling into the disconnect issue. My focus is on the frontend's inability to recognize the backend's "disconnected" state post-restart. I'm trying to pinpoint where the communication breaks down – is it a messaging problem, a timing glitch, or a UI update failure? I need to clarify what the user's expected 'disconnected' behavior actually is, and if there are any specific messages or events I can track.

### Isolating the Root Cause

I'm making progress, focusing on how the UI reacts to the backend restart. The logs show a successful restart command, but the UI remains "connected." I've reviewed the `navigation.html` file, which includes the disconnect logic, and am trying to understand the interplay between the `restoreConnectionUI` function and the backend's "disconnected" signal. I'm checking if the UI is correctly interpreting and acting on the `disconnected: true` signal after the restart.

### Pinpointing the Communication Gap

I've discovered the root cause is the Navigation bar's lack of awareness of the "Restart" action. Clicking the Navigation bar manually correctly handles the disconnection, but the Power page's restart API call doesn't notify the Navigation bar. The UI assumes "connected" as `localStorage` data persists. I'm focusing on two solutions: an event-driven approach where the Power page signals the Navigation bar, or a less efficient polling method. I'm leaning toward the "Push" approach.

### Addressing the UI Mismatch

I've circled back to the user's focus on the navigation bar. The crucial disconnect function in `navigation.html` is only triggered by the "Disconnect" button, not the backend restart initiated from the Power page. I'm now zeroing in on how to best update the navigation bar *without* modifying the Power page logic. It seems clear the existing disconnect event is triggered only by the navigation button. I need to understand the relationship between `localStorage` and this state, and if an event listener can make the necessary update.

### Implementing Remote Disconnect

I'm zeroing in on a solution. The crux is that the 'Restart' command from the Power page doesn't automatically trigger the Navigation bar's disconnect process. My focus is on making the disconnect logic accessible, so it can be called from other parts of the application. I'm considering two approaches. Either I can refactor the Navigation bar's code to extract the disconnect logic into a globally accessible function, or I can leverage a custom event listener for a more elegant approach. The self-contained approach seems cleaner.

### Integrating UI Disconnect

I'm focusing on cleaning up the UI following a restart command. It seems the disconnect button's logic is only triggered on click, not programmatically. I've developed a solution for the `navigation.html` file, by creating a `handleDisconnectUI()` function. I've made this function globally accessible, so it can be called from the "Restart" button's success callback. This allows the backend to handle its restart and trigger the required frontend UI updates. It's cleaner to handle UI disconnect outside the initial action.

### Creating Global UI Function

I'm focused on finalizing the UI updates for the restart process. I'm extracting the UI reset logic from `navigation.html` into a global function, `window.resetConnectionUI()`. This will be called from the "Restart" button's callback in the `power.html` file, ensuring the UI accurately reflects the disconnected state following a restart command, triggered via the `disconnect-server` API. This will simplify updates and better manage state.

### Implementing Global Disconnect

I've crafted a global function, `window.handleClientDisconnect()`, in `navigation.html`. This neatly encapsulates the UI reset logic. Now, other pages, like `power.html`, can call it directly after the restart API call, ensuring the UI correctly reflects the disconnected state by removing local storage entries, hiding the disconnect button, and showing the connect button. I'm excited by this simplification.

### Simplifying Disconnection Flow

I've established a mechanism for automatic UI updates after a disconnect. I've created the `triggerDisconnectUI()` function, which cleans up local storage and the UI. This is now hooked to the existing "Disconnect" button, and will be easily accessible from other parts of the application, such as the Power page, by adding a simple check on the API response. I'm focusing on providing clear instructions on its implementation within relevant callbacks.

Vấn đề nằm ở chỗ: **Logic "đọn dẹp giao diện" (xóa localStorage, đổi nút Disconnect thành Connect) hiện tại đang bị nhốt (encapsulated) bên trong sự kiện click của nút Disconnect.**

Khi bạn bấm nút **Restart** ở trang Power:

1. Lệnh gửi lên Server Python thành công (`200 OK`).
2. Server Python đã ngắt kết nối socket (như code bạn sửa ở bước trước).
3. Tuy nhiên, trang web (Javascript trên Navigation bar) **không hề biết** việc này vừa xảy ra. Nó vẫn nghĩ là đang kết nối vì chưa ai bảo nó xóa `localStorage` hay đổi lại giao diện cả.

Để sửa lỗi này, bạn cần **tách logic dọn dẹp giao diện ra thành một hàm dùng chung (Global Function)** để có thể gọi nó từ bất cứ đâu (kể cả từ trang Power sau khi Restart thành công).

Dưới đây là cách sửa file `navigation.html`:

### Bước 1: Sửa file `navigation.html`

Bạn hãy thay thế đoạn script xử lý `disconnectBtn` cũ bằng đoạn code đã được tối ưu hóa này. Tôi đã tách phần xử lý giao diện ra thành hàm `window.handleClientDisconnect`.

```
<script>
    document.addEventListener('DOMContentLoaded', async function() {
        const serverSelect = document.getElementById('server-select');
        const connectBtn = document.getElementById('connect-server-btn');
        const disconnectBtn = document.getElementById('disconnect-server-btn');
        // ... các biến khác giữ nguyên ...

        // ... (Giữ nguyên phần check server restart và các hàm save/get connection)
        ...

        // ===== [MỚI] HÀM DỌN DẸP GIAO DIỆN TOÀN CỤC
        =====
        // Hàm này được gán vào window để các trang khác (như Power) có thể gọi được
        window.handleClientDisconnect = function() {
            console.log('⌚ Performing UI Disconnect...');

            // 1. Xóa dữ liệu trong localStorage
            localStorage.removeItem('connectedServer');
            localStorage.removeItem('connectionTime');
            localStorage.removeItem('ACTIVE_STREAM_MODE'); // Xóa cờ đang stream nếu
            có
            sessionStorage.clear();

            // 2. Cập nhật giao diện nút bấm
            if (connectBtn && disconnectBtn && serverSelect) {
                // Reset Select Box
                serverSelect.innerHTML = '<option value="" disabled selected>Select
                Server</option>';

                // Hiện nút Connect, ẩn nút Disconnect
                connectBtn.classList.remove('hidden', 'bg-green-600', 'dark:bg-green-
                500');
                connectBtn.classList.add('bg-primary-700', 'dark:bg-primary-600');
                connectBtn.textContent = 'Connect';
                disconnectBtn.classList.add('hidden');

                // Enable lại nút Connect (để phòng đang bị disable)
                connectBtn.disabled = false;
            }

            // 3. (Tùy chọn) Reload lại trang Home để reset sạch sẽ các view
            // Nếu đang dùng HTMX
        }
    });

```

```
if (typeof htmx !== 'undefined') {
    htmx.ajax('GET', '/remote/home/', '#main-content');
}

// ===== LOGIC NÚT DISCONNECT (ĐÃ SỬA) =====
if (disconnectBtn) {
    disconnectBtn.addEventListener('click', async function() {

        // Check Stream (Giữ nguyên logic bảo vệ cũ của bạn)
        const activeStream = localStorage.getItem('ACTIVE_STREAM_MODE');
        if (activeStream) {
            showNotification("warning", `⚠ Cannot disconnect while
${activeStream} is active.`);
            return;
        }

        disconnectBtn.disabled = true;
        disconnectBtn.textContent = 'Disconnecting...';

        try {
            // Gọi API báo Server Python ngắt socket
            const response = await fetch('/remote/api/disconnect-server/', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json',
                    'X-CSRFToken': getCookie('csrftoken')
                }
            });

            const data = await response.json();

            if (data.success) {
                showNotification("success", "🔌 Disconnected successfully.");

                // GỌI HÀM DỌN DẸP GIAO DIỆN VỪA TẠO
                window.handleClientDisconnect();

            } else {
                showNotification("error", data.message || 'Failed to disconnect.');
                // Nếu lỗi API, vẫn trả lại trạng thái nút bấm
                disconnectBtn.disabled = false;
                disconnectBtn.textContent = 'Disconnect';
            }
        } catch (error) {
            showNotification("error", 'Disconnect error: ' + error.message);
            disconnectBtn.disabled = false;
            disconnectBtn.textContent = 'Disconnect';
        }
    });
}

// ... (Các phần code connectBtn, discoverBtn giữ nguyên) ...
```

```
});  
</script>
```

## Bước 2: Kích hoạt từ trang Power

Bây giờ, ở file HTML hoặc Javascript xử lý nút Restart/Shutdown (file bạn chưa gửi nhưng tôi đoán logic của nó), bạn cần thêm một dòng lệnh để gọi hàm `window.handleClientDisconnect()` khi nhận được phản hồi thành công.

Ví dụ logic xử lý nút Restart của bạn sẽ trông như thế này:

```
// Ví dụ trong file power.html hoặc file JS xử lý nút Restart  
async function sendPowerCommand(action) {  
    try {  
        const response = await fetch(`/remote/api/power/${action}/`, { method:  
        'POST', ... });  
        const data = await response.json();  
  
        if (data.status === 'success') {  
            alert("Command Sent: " + data.message);  
  
            // === THÊM DÒNG NÀY ===  
            // Kiểm tra nếu Server báo đã ngắt kết nối (flag 'disconnected' ta  
thêm ở bài trước)  
            if (data.disconnected === true) {  
                // Gọi hàm dọn dẹp giao diện ở Navigation bar  
                if (window.handleClientDisconnect) {  
                    window.handleClientDisconnect();  
                }  
            }  
            // ======  
        }  
    } catch (e) {  
        console.error(e);  
    }  
}
```

## Tóm lại:

1. **Frontend (navigation.html):** Tạo hàm `window.handleClientDisconnect` để reset UI về trạng thái chưa kết nối.
2. **Backend (views.py):** Đã trả về `{"disconnected": true}` (đã làm ở bước trước).
3. **Frontend (power logic):** Khi nhận `disconnected: true`, gọi ngay `window.handleClientDisconnect()`.

Như vậy, ngay khi bấm Restart, giao diện sẽ lập tức quay về màn hình "Connect", đồng bộ hoàn toàn với trạng thái thực tế của Server.

## Prompt:

## power\_partial - HTML

Đây có phải file bạn muốn không?

Response:

Thinking:

Thinking:

### Examining Auto-Disconnect

I've been drilling into the user's request: implementing an "Auto-disconnect" feature in the frontend following "Shutdown" or "Restart" commands. My focus is on how the frontend can be made to handle this disconnection gracefully, especially considering the potential for incomplete data transfers during these events. I am working on the best way to handle this.

### Pinpointing Disconnect Logic

I've determined that the provided `power_partial.html` is the correct location for integrating the disconnection logic. My analysis confirms this file houses the shutdown and restart button functionality. Now I'm focused on where and how to insert the check to trigger the frontend disconnect, considering the backend signal. I am thinking this is most likely where the `apiCall` is made, to ensure the disconnect happens after the server acknowledges the command.

### Incorporating Disconnect Check

I am now focusing on integrating the `disconnected` flag from the API response into the shutdown and restart functions. I'll insert a check within the `if (res.status === 'success')` block to trigger `window.handleClientDisconnect()` if the `disconnected` flag is true. A slight delay after the success message seems optimal to allow users to register the message before the disconnect. This design aligns with previous user requests for an automatic disconnect after server commands.

### Fine-tuning Disconnect Delay

I'm now refining the timeout for the automatic disconnect. I've decided on a 1.5-second delay to strike a balance between a smooth user experience and prompt action. The code now includes an update to the UI message, indicating that a disconnect is in progress, right before triggering `handleClientDisconnect()` via `setTimeout`. This will give a better UX, as the user will know something is happening, and is a slight improvement over the direct call.

### Implementing Auto-Disconnect

I'm now integrating the backend's `disconnected` flag into the `performShutdown` and `performRestart` functions. The aim is to bridge the server's response with the frontend's cleanup mechanism. My plan is to add a check for `res.disconnected` and trigger `window.handleClientDisconnect()` with a 1.5-second delay for a smoother user experience. This follows my previous steps, which will avoid the glitchy feel, and give the user a chance to see that the command went through.

## Chính xác! Đây đúng là file cần sửa.

Bạn chỉ cần sửa đoạn script bên dưới của file `power_partial.html` để kiểm tra cờ `disconnected` mà Backend gửi về.

Tôi sẽ thêm một chút thời gian trễ (**setTimeout 1.5 giây**) để người dùng kịp nhìn thấy thông báo "Thành công" màu xanh trước khi giao diện tự động reset về màn hình Connect. Điều này giúp trải nghiệm người dùng mượt mà hơn.

Dưới đây là đoạn code đã sửa cho phần `<script>`:

```
<script>
    // Shutdown
    async function performShutdown() {
        if (!confirm('⚠ Bạn có chắc muốn SHUTDOWN máy client không?\n\nMáy sẽ
tắt ngay lập tức!')) {
            return;
        }

        const result = document.getElementById('shutdownResult');
        result.className = 'mt-4 text-sm text-center p-3 rounded';
        result.innerHTML = 'Đang gửi lệnh shutdown...';
        result.classList.remove('hidden');

        // Gọi API (Backend đã được sửa để trả về {disconnected: true})
        const res = await apiCall('/remote/api/power/shutdown/', 'POST');

        if (res.status === 'success') {
            result.className = 'mt-4 text-sm text-center p-3 rounded bg-green-200
dark:bg-green-700 text-green-900 dark:text-green-100';
            result.innerHTML = `☑ ${res.message} || 'Shutdown thành công.'`;

            // === [THÊM MỚI] KIỂM TRA CỜ DISCONNECT ===
            if (res.disconnected === true) {
                result.innerHTML += '<br>⚠ Đang ngắt kết nối giao diện...';

                // Đợi 1.5 giây để người dùng đọc thông báo rồi mới reset
                setTimeout(() => {
                    if (window.handleClientDisconnect) {
                        window.handleClientDisconnect();
                    }
                }, 1500);
            }
            // =====
        } else {
            result.className = 'mt-4 text-sm text-center p-3 rounded bg-red-200
dark:bg-red-700 text-red-900 dark:text-red-100';
            result.innerHTML = `☒ ${res.message} || 'Lỗi khi gửi lệnh
shutdown'`;
        }
    }
</script>
```

```
// Restart
async function performRestart() {
    if (!confirm('⚠ Bạn có chắc chắn muốn RESTART máy client không?\n\nMáy sẽ
    khởi động lại ngay lập tức!')) {
        return;
    }

    const result = document.getElementById('restartResult');
    result.className = 'mt-4 text-sm text-center p-3 rounded';
    result.innerHTML = 'Đang gửi lệnh restart...';
    result.classList.remove('hidden');

    // Gọi API
    const res = await apiCall('/remote/api/power/restart/', 'POST');

    if (res.status === 'success') {
        result.className = 'mt-4 text-sm text-center p-3 rounded bg-green-200
dark:bg-green-700 text-green-900 dark:text-green-100';
        result.innerHTML = `✅ ${res.message} || 'Restart thành công.'`;

        // === [THÊM MỚI] KIỂM TRA CỜ DISCONNECT ===
        if (res.disconnected === true) {
            result.innerHTML += '<br>⚠ Đang ngắt kết nối giao diện...';

            // Đợi 1.5 giây để người dùng đọc thông báo rồi mới reset
            setTimeout(() => {
                if (window.handleClientDisconnect) {
                    window.handleClientDisconnect();
                }
            }, 1500);
        }
        // =====
    } else {
        result.className = 'mt-4 text-sm text-center p-3 rounded bg-red-200
dark:bg-red-700 text-red-900 dark:text-red-100';
        result.innerHTML = `✖ ${res.message} || 'Lỗi khi gửi lệnh restart'`;
    }
}
</script>

<script>
// Common API call function
async function apiCall(url, method='GET', body=null) {
    const options = {
        method: method,
        headers: {'Content-Type': 'application/json'}
    };
    if (body) options.body = JSON.stringify(body);

    try {
        const res = await fetch(url, options);
        return await res.json();
    }
}
```

```
    } catch (err) {
      console.error(err);
      return { status: 'error', message: 'Connection Error' };
    }
  }
</script>
```

Giải thích thay đổi:

1. **Kiểm tra res.disconnected === true**: Backend (views.py) gửi cờ này khi lệnh shutdown/restart thành công.
2. **Gọi window.handleClientDisconnect()**: Đây là hàm chúng ta vừa viết ở navigation.html để xóa localStorage và đổi nút Disconnect thành Connect.
3. **setTimeout(..., 1500)**: Tôi thêm độ trễ 1.5 giây. Nếu không có cái này, ngay khi bấm nút, giao diện sẽ "chớp" một cái và quay về màn hình Connect ngay lập tức, người dùng sẽ không kịp thấy dòng chữ " Shutdown thành công".

Prompt:

Ok, bây giờ tôi muốn thêm 1 tính năng mới:

Trang home của website, khi connect vào 1 server, tôi muốn lấy những thông tin của máy chủ server đó (như là CPU load, RAM usage, network latency, battery và các thông tin như Hostname, internal IP, OS...) Nói chung là chỉ cần những thông tin nào có thể được lấy một cách đơn giản, không cần quá phức tạp.

Khi disconnect thì tự động tắt hết những thông tin này: nghĩa là trong home page sẽ có 1 cái cửa sổ sẽ chỉ display khi đang kết nối vào 1 server nào đó.

Đây là những file mà tôi nghĩ sẽ quan trọng: views.py, urls.py, socket\_client\_persistent, server.cs, home.html, home\_partial.html

Đây là file home.html hiện tại:

```
{% extends "layouts/base.html" %}

{% load static %}

{% block content %}

{% include "remote_control/partials/applications_partial.html" %}

{% endblock content %}
```

Cái applications\_partial.html là dummy thôi

home\_partial thì còn trống nhé, viết thoải mái.

Bạn cần hoặc quên file nào thì cứ nói, tôi sẽ gửi. Đừng đoán code.

Response:

Thinking:

Thinking:

## Reviewing the Project Scope

I've been analyzing the core requirements for the system dashboard. Specifically, I'm focusing on the display of crucial system information, like CPU usage, RAM utilization, and network latency, for authenticated users. The initial outline is forming, and the design needs to be consistent with the other parts of the site.

## Gathering System Metrics

I'm now fully immersed in the system metrics gathering. The C# server side is demanding attention, and I'm developing a new command handler to gather and send key system information back to the client. I've pinpointed the necessary classes for fetching CPU, RAM, battery, OS, hostname, and IP data. The plan is to package this information into a JSON format for transmission.

## Deciding Data Transmission Format

I've been re-evaluating the data transmission format. Initially, I leaned toward delimited strings, but after considering data type complexities and dependency concerns, I'm now leaning toward a manually constructed JSON string. It's a robust and flexible solution for mixed data, and importantly, avoids external library dependencies in the current project environment. The alternative of a simple pipe-delimited string would be usable too, but JSON seems to better future-proof the design.

## Defining API Endpoints

I'm now integrating the backend and frontend. I've defined a new API endpoint, `api/server/stats/`, in `urls.py` and written the corresponding view function in `views.py`. This view function will call the client method to retrieve the system information and return it as JSON. I'm also adding a method, `get_system_info()`, in the existing client code to send the `SYSTEM_INFO` command, receive the data, and parse it into a Python dictionary. I'll need to measure the latency to return it with the data.

## Constructing System Info Retrieval

I'm now outlining a step-by-step implementation plan. I'll begin by modifying `server.cs`, specifically adding a new case for "SYSTEM\_INFO" in `HandleClientCommunication` and creating a helper function, `GetSystemInfo()`, to retrieve relevant system data. This helper will gather the OS name, hostname, IP, CPU percentage, available and total RAM, and battery status. I've decided on line-separated strings for now to reduce dependency on external C# JSON libraries, making parsing in Python easier. I will also incorporate the necessary `PerformanceCounter` checks in C# to ensure full compatibility. Next, I'll update the Python client, `socket_client_persistent.py`, to include a function `get_system_info()` and measure the latency.

## Developing the Frontend Dashboard

I'm now fully immersed in the frontend dashboard development. The plan is to create a new Django view and URL, along with a corresponding Tailwind CSS-based dashboard section in

`home_partial.html`. I'll integrate JavaScript to fetch and update the system metrics every few seconds. I've also planned for a "Not Connected" state, ensuring the dashboard handles cases when data retrieval fails. The current focus is ensuring the system can properly respond to frontend requests.

## Improving Metric Collection

I'm now refining the data collection process. I'm focusing on RAM information. While `PerformanceCounter` provides available RAM, getting total RAM is complex. I'll utilize WMI via `ManagementObjectSearcher` for total RAM and OS name, caching for efficiency. I'll explore newer .NET methods like `GC.GetGCMemoryInfo` if WMI proves slow.

## Refining Data Gathering

I'm now focusing on system metric reliability, particularly avoiding external dependencies. I'm exploring non-WMI approaches to fetch total RAM in C#, using P/Invoke or `Microsoft.VisualBasic` as standard options. I'm now leaning towards showing available RAM only, for simplicity, and CPU load via `PerformanceCounter`. This minimizes potential user configuration issues, and is in line with the "simple" requirements of the design. I'll provide a WMI snippet, but it will be optional. I've successfully incorporated standard `PerformanceCounter` checks in C# to ensure full compatibility.

## Clarifying Data Requirements

I'm now revising my approach to the core system metrics. I'm choosing to prioritize simplicity and minimize external dependencies. I've decided to display CPU load and available RAM, obtained via `PerformanceCounter`, for reliability. I'll construct a C# helper, using `PerformanceCounter`, to fetch this data.

Đây là một tính năng rất thiết thực. Để làm được điều này, chúng ta cần đi từ Server (C#) để thu thập dữ liệu -> Client (Python) để nhận và tính toán độ trễ -> View (Django) để trả API -> Frontend (HTML/JS) để hiển thị.

Tôi sẽ đi từng bước, chỉnh sửa các file cần thiết.

### 1. Cập nhật `server.cs` (ServerApp)

Chúng ta cần thêm module `SYSTEM_INFO` để trả về các thông số máy tính.

**Lưu ý:** Để lấy % CPU chính xác, ta cần dùng `PerformanceCounter`. Bạn hãy thêm đoạn code này vào trong class `server`:

```
// --- THÊM VÀO PHẦN KHAI BÁO BIẾN TOÀN CỤC (đầu class server) ---
PerformanceCounter cpuCounter;
PerformanceCounter ramCounter;

// --- CẬP NHẬT CONSTRUCTOR (public server()) ---
public server()
{
    InitializeComponent();
    CheckForIllegalCrossThreadCalls = false;
    this.FormClosing += new FormClosingEventHandler(server_FormClosing);
```

```
// Khởi tạo Counter cho CPU và RAM
try {
    cpuCounter = new PerformanceCounter("Processor", "% Processor Time",
"_Total");
    ramCounter = new PerformanceCounter("Memory", "Available MBytes");
} catch { }

// ... (phần xóa log cũ giữ nguyên)
}

// --- CẬP NHẬT HÀM HandleClientCommunication ---
// Thêm case "SYSTEM_INFO" vào switch(s)
switch (s)
{
    // ... các case cũ giữ nguyên ...

    case "SYSTEM_INFO":
        send_system_info();
        break;

    // ...
}

// --- THÊM HÀM MỚI Ở CUỐI CLASS ---
public void send_system_info()
{
    try
    {
        // 1. CPU Load
        // Lần gọi đầu tiên của NextValue() thường trả về 0, nên gọi 2 lần hoặc
        chấp nhận delay
        // Ở đây ta gọi 1 lần, nếu loop liên tục từ client thì nó sẽ chính xác dần
        float cpu = cpuCounter.NextValue();

        // 2. RAM Available
        float ram = ramCounter.NextValue();

        // 3. Battery Info (Yêu cầu Reference: System.Windows.Forms - đã có sẵn)
        PowerStatus pwr = SystemInformation.PowerStatus;
        String battery = (pwr.BatteryLifePercent * 100).ToString() + "%";
        if (pwr.PowerLineStatus == PowerLineStatus.Online) battery += "
(Charging)";

        // 4. Basic Info
        String hostname = Dns.GetHostName();
        String os = Environment.OSVersion.ToString();

        // Lấy IP nội bộ (IPv4)
        String ipAddr = "Unknown";
        try {
            var host = Dns.GetHostEntry(Dns.GetHostName());
            foreach (var ip in host.AddressList) {
                if (ip.AddressFamily == AddressFamily.InterNetwork) {

```

```
        ipAddr = ip.ToString();
        break;
    }
}
} catch {}

// Gửi về dạng chuỗi phân cách bởi ký tự "|"
// Format: CPU|RAM_FREE|BATTERY|HOSTNAME|OS|IP
String response = $"{cpu:0.0}|{ram}|{battery}|{hostname}|{os}|{ipAddr}";

Program.nw.WriteLine(response);
Program.nw.Flush();
}
catch (Exception ex)
{
    Program.nw.WriteLine("ERROR| " + ex.Message);
    Program.nw.Flush();
}
}
```

## 2. Cập nhật `socket_client_persistent.py`

Thêm hàm để gọi lệnh trên và tính toán độ trễ (Latency).

```
# --- THÊM VÀO class PersistentRemoteClient ---

def get_system_stats(self):
    """
    Lấy thông tin hệ thống và tính Ping (Latency)
    """
    if not self.connected:
        return {"status": "error", "message": "Not connected"}

    with self._lock:
        try:
            start_time = time.time() # Bắt đầu đo giờ

            self._send_str("SYSTEM_INFO")
            response = self._recv_line()

            end_time = time.time() # Kết thúc đo giờ

            # Tính latency (ms)
            latency = round((end_time - start_time) * 1000, 2)

            if response.startswith("ERROR"):
                return {"status": "error", "message": response}

            # Parse dữ liệu: CPU|RAM_FREE|BATTERY|HOSTNAME|OS|IP
            parts = response.split('|')
```

```

        data = {
            "cpu_load": parts[0] if len(parts) > 0 else "0",
            "ram_free": parts[1] if len(parts) > 1 else "0",
            "battery": parts[2] if len(parts) > 2 else "Unknown",
            "hostname": parts[3] if len(parts) > 3 else "Unknown",
            "os_info": parts[4] if len(parts) > 4 else "Unknown",
            "internal_ip": parts[5] if len(parts) > 5 else "Unknown",
            "latency": latency
        }

        self._send_str("QUIT")
        return {"status": "success", "data": data}

    except Exception as e:
        return {"status": "error", "message": str(e)}

```

### 3. Cập nhật `views.py`

Tạo API endpoint trả về JSON cho Frontend.

```

# --- THÊM VÀO views.py ---
@require_http_methods(["GET"])
def get_server_stats(request):
    """API: Lấy thông số Dashboard (CPU, RAM, Ping...)"""
    client = _get_client(request)

    # Nếu chưa kết nối, trả về status đặc biệt để Frontend ẩn Dashboard
    if not client or not client.connected:
        return JsonResponse({
            "status": "disconnected",
            "message": "Not connected"
        })

    result = client.get_system_stats()
    return JsonResponse(result)

```

### 4. Cập nhật `urls.py`

Đăng ký đường dẫn cho API mới.

```

urlpatterns = [
    # ... các đường dẫn cũ ...

    # API Dashboard Stats
    path('api/stats/', views.get_server_stats, name='api_server_stats'),
]

```

## 5. Cập nhật [home.html](#)

Sửa lại file này để include cái partial mới của chúng ta.

```
{% extends "layouts/base.html" %}  
{% load static %}  
  
{% block content %}  
  
    <div id="dashboard-monitor-container" class="mb-6">  
        {% include "remote_control/partials/home_partial.html" %}  
    </div>  
  
    <div id="other-widgets">  
        {% include "remote_control/partials/applications_partial.html" %}  
    </div>  
  
{% endblock content %}
```

## 6. Tạo nội dung [home\\_partial.html](#) (QUAN TRỌNG)

File này sẽ chứa giao diện Dashboard đẹp mắt và Script tự động fetch dữ liệu mỗi 2 giây.

```
<div id="live-dashboard" class="hidden transition-opacity duration-500 ease-in-out">  
  
    <div class="mb-4 bg-white border border-gray-200 rounded-lg shadow-sm dark:border-gray-700 dark:bg-gray-800 p-4">  
        <div class="flex flex-wrap justify-between items-center">  
            <div>  
                <h2 class="text-xl font-bold text-gray-900 dark:text-white flex items-center gap-2">  
                    <span class="w-3 h-3 bg-green-500 rounded-full animate-pulse"></span>  
                    Connected: <span id="dash-hostname" class="text-primary-600 dark:text-primary-400">Loading...</span>  
                </h2>  
                <p class="text-sm text-gray-500 dark:text-gray-400 mt-1">  
                    IP: <span id="dash-ip">...</span> | OS: <span id="dash-os">...</span>  
                </p>  
            </div>  
            <div class="text-right">  
                <div class="text-sm font-medium text-gray-900 dark:text-white">  
                    Latency: <span id="dash-latency" class="text-green-500 font-bold">--</span> ms  
                </div>  
                <div class="text-xs text-gray-500">Last update: <span id="dash-last-update">--::--</span></div>  
            </div>  
        </div>  
    </div>
```

```
</div>
</div>

<div class="grid grid-cols-1 md:grid-cols-3 gap-4">

    <div class="p-4 bg-white border border-gray-200 rounded-lg shadow-sm dark:bg-gray-800 dark:border-gray-700">
        <div class="flex items-center justify-between mb-2">
            <h3 class="text-base font-medium text-gray-500 dark:text-gray-400">CPU Load</h3>
                <svg class="w-6 h-6 text-gray-400" fill="none" stroke="currentColor" viewBox="0 0 24 24"><path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M9 3v2m6-2v2M9 19v2m6-2v2M5 9H3m2 6H3m18-6h-2m2 6h-2M7 19h10a2 2 0 002-2V7a2 2 0 00-2-2H7a2 2 0 00-2 2v10a2 2 0 002 2zM9 9h6v6H9V9z"></path></svg>
        </div>
        <div class="flex items-end gap-2">
            <span id="dash-cpu" class="text-3xl font-bold text-gray-900 dark:text-white">0</span>
            <span class="text-lg font-medium text-gray-500 mb-1">%</span>
        </div>
        <div class="w-full bg-gray-200 rounded-full h-2.5 dark:bg-gray-700 mt-2">
            <div id="dash-cpu-bar" class="bg-blue-600 h-2.5 rounded-full transition-all duration-500" style="width: 0%"></div>
        </div>
    </div>

    <div class="p-4 bg-white border border-gray-200 rounded-lg shadow-sm dark:bg-gray-800 dark:border-gray-700">
        <div class="flex items-center justify-between mb-2">
            <h3 class="text-base font-medium text-gray-500 dark:text-gray-400">RAM Available</h3>
                <svg class="w-6 h-6 text-gray-400" fill="none" stroke="currentColor" viewBox="0 0 24 24"><path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M19.428 15.428a2 2 0 00-1.022-.547l-2.387-.477a6 6 0 00-3.86.517l-.318.158a6 6 0 01-3.86.517L6.05 15.21a2 2 0 00-1.806.547M8 4h8l-1 1v5.172a2 2 0 00.586 1.414l5 5c1.26 1.26.367 3.414-1.415 3.414H4.828c-1.782 0-2.674-2.154-1.414-3.414l5-5a2 2 0 00.009 10.172V5L8 4z"></path>
            </div>
            <div class="flex items-end gap-2">
                <span id="dash-ram" class="text-3xl font-bold text-gray-900 dark:text-white">0</span>
                <span class="text-lg font-medium text-gray-500 mb-1">MB</span>
            </div>
            <p class="text-xs text-gray-500 mt-2">Free memory available</p>
        </div>

        <div class="p-4 bg-white border border-gray-200 rounded-lg shadow-sm dark:bg-gray-800 dark:border-gray-700">
            <div class="flex items-center justify-between mb-2">
                <h3 class="text-base font-medium text-gray-500 dark:text-gray-400">Battery</h3>
```

```
        <svg class="w-6 h-6 text-gray-400" fill="none"
stroke="currentColor" viewBox="0 0 24 24"><path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M13 10V3L4 14h7v7l9-11h-7z"></path></svg>
    </div>
    <div class="flex items-end gap-2">
        <span id="dash-battery" class="text-3xl font-bold text-gray-900
dark:text-white">--</span>
    </div>
    <p class="text-xs text-gray-500 mt-2">Power Status</p>
</div>

</div>
</div>

<div id="monitor-placeholder" class="text-center p-8 bg-gray-50 border border-
dashed border-gray-300 rounded-lg dark:bg-gray-800 dark:border-gray-700">
    <svg class="w-12 h-12 mx-auto text-gray-400 mb-4" fill="none"
stroke="currentColor" viewBox="0 0 24 24"><path stroke-linecap="round" stroke-
linejoin="round" stroke-width="2" d="M12 6v6m0 0v6m0-6h6m-6 0H6"></path></svg>
    <h3 class="text-lg font-medium text-gray-900 dark:text-white">No Active
Connection</h3>
    <p class="text-gray-500 dark:text-gray-400">Connect to a server to view system
performance.</p>
</div>

<script>
    document.addEventListener("DOMContentLoaded", function() {
        const dashboard = document.getElementById('live-dashboard');
        const placeholder = document.getElementById('monitor-placeholder');
        let updateInterval = null;

        // Hàm fetch dữ liệu
        async function fetchStats() {
            try {
                const response = await fetch('/remote/api/stats/');
                // Nếu trả về 404 hoặc lỗi mạng, coi như disconnect
                if (!response.ok) throw new Error('Network error');

                const data = await response.json();

                // Kiểm tra xem backend báo disconnected chưa
                if (data.status === 'disconnected') {
                    stopMonitoring();
                    return;
                }

                if (data.status === 'success' && data.data) {
                    showDashboard(data.data);
                }
            } catch (error) {
                console.log("Monitoring paused (error/disconnect)");
                // Nếu lỗi liên tục (vd server tắt), ta có thể chọn stop hoặc
                retry
            }
        }
    })
</script>
```

```
        }

    }

    function showDashboard(data) {
        // Hiện dashboard, ẩn placeholder
        dashboard.classList.remove('hidden');
        placeholder.classList.add('hidden');

        // Đèn dữ liệu
        document.getElementById('dash-hostname').textContent = data.hostname;
        document.getElementById('dash-ip').textContent = data.internal_ip;
        document.getElementById('dash-os').textContent = data.os_info;
        document.getElementById('dash-latency').textContent = data.latency;

        // Update CPU
        const cpuVal = parseFloat(data.cpu_load);
        document.getElementById('dash-cpu').textContent = cpuVal;
        document.getElementById('dash-cpu-bar').style.width =
` ${Math.min(cpuVal, 100)}%`;

        // Colorize CPU bar
        const bar = document.getElementById('dash-cpu-bar');
        if(cpuVal > 80) bar.className = "bg-red-600 h-2.5 rounded-full transition-all duration-500";
        else if(cpuVal > 50) bar.className = "bg-yellow-500 h-2.5 rounded-full transition-all duration-500";
        else bar.className = "bg-blue-600 h-2.5 rounded-full transition-all duration-500";

        // Update RAM
        document.getElementById('dash-ram').textContent = data.ram_free;

        // Update Battery
        document.getElementById('dash-battery').textContent = data.battery;

        // Update Time
        const now = new Date();
        document.getElementById('dash-last-update').textContent =
now.toLocaleTimeString();
    }

    function stopMonitoring() {
        dashboard.classList.add('hidden');
        placeholder.classList.remove('hidden');
        if (updateInterval) {
            clearInterval(updateInterval);
            updateInterval = null;
        }
    }

    function startMonitoring() {
        // Fetch ngay lập tức
        fetchStats();
        // Sau đó loop mỗi 2 giây
    }
}
```

```

        if (!updateInterval) {
            updateInterval = setInterval(fetchStats, 2000);
        }
    }

    // --- LOGIC KÍCH HOẠT ---
    // 1. Kiểm tra localStorage xem có đang connect không
    const connectedServer = localStorage.getItem('connectedServer');
    if (connectedServer) {
        startMonitoring();
    }

    // 2. Lắng nghe sự kiện click nút Connect (để bắt đầu monitor ngay)
    const connectBtn = document.getElementById('connect-server-btn');
    if(connectBtn) {
        connectBtn.addEventListener('click', () => {
            // Đợi 1 xíu để kết nối thiết lập xong rồi mới chạy monitor
            setTimeout(startMonitoring, 1000);
        });
    }

    // 3. Lắng nghe sự kiện click nút Disconnect (để dừng monitor ngay)
    const disconnectBtn = document.getElementById('disconnect-server-btn');
    if(disconnectBtn) {
        disconnectBtn.addEventListener('click', stopMonitoring);
    }
});
</script>

```

Cách hoạt động:

1. **Khi trang load:** Script trong `home_partial.html` kiểm tra `localStorage`. Nếu thấy key `connectedServer` (đã connect), nó sẽ ẩn placeholder và hiện Dashboard, đồng thời gọi API mỗi 2 giây.
2. **API:** Gọi xuống C#, C# dùng `PerformanceCounter` lấy CPU/RAM và trả về chuỗi.
3. **Client:** Nhận chuỗi, tính thời gian `recv - send` để ra Ping, rồi trả JSON về frontend.
4. **Khi Disconnect:** API trả về `status: disconnected`, Script JS nhận thấy và tự động ẩn Dashboard đi.

Prompt:

Khi vào web, connect xong rồi thì home vẫn ghi thế này:

No Active Connection

Connect to a server to view system performance.

Log bên Python:

[16/Dec/2025 14:27:57] "GET /remote/api/discover-servers/ HTTP/1.1" 200 141

[16/Dec/2025 14:27:59] "POST /remote/api/connect-server/ HTTP/1.1" 200 90

[16/Dec/2025 14:28:00] "GET /remote/home/ HTTP/1.1" 200 9217

[16/Dec/2025 14:28:04] "GET /remote/applications/ HTTP/1.1" 200 14421

[16/Dec/2025 14:28:04] "GET /remote/home/ HTTP/1.1" 200 9217

[16/Dec/2025 14:28:08] "GET /remote/processes/ HTTP/1.1" 200 14139

Tôi đã làm tất cả những gì bạn nói

## Response:

Thinking:

Thinking:

### Diagnosing Connection Issue

I've been analyzing the user's reported problem. The initial log indicates a successful server connection, which is odd. My focus is now on why the "No Active Connection" placeholder persists despite the connection's success. I'm suspecting a mismatch between the reported connection status and the actual state. Perhaps a client-side issue is preventing the updated state from being reflected, so I'll explore how the client handles the connection signal.

### Isolating the Root Cause

I'm now zeroing in on why the stats aren't being fetched. My recent dive into the code revealed a potential flaw. The script relies on `DOMContentLoaded`, which might not fire when content is dynamically updated in modern frameworks, which are used by the user. I'll focus my investigation on how the home page's HTML gets updated, which will likely point to the missing signal for the stats to be fetched, specifically, the `fetchStats` function. I'll need to figure out how to initialize the stats fetches in such an event-driven framework.

### Identifying Execution Points

I've realized the primary script's execution isn't triggered on subsequent page loads via SPA or HTMX. I've noted that `DOMContentLoaded` has already fired, rendering the existing setup useless. I'm focusing on the user's visit to `/remote/home/` to correlate `connectedServer` presence. This reinforces my hypothesis on why the script might not be running correctly.

### Analyzing Javascript Execution

I've realized the `DOMContentLoaded` event listener on `home_partial.html` is the culprit. When the user navigates using HTMX, it adds a callback to an event that has already occurred. This won't initialize the connection properly. I'm focusing on the `home_partial.html` script. The most reliable fix is to call the initialization function immediately. If the DOM is ready, call `setupConnection`, otherwise, add it to `DOMContentLoaded`. This hybrid approach ensures the script executes on both full and partial loads.

### Simplifying the Connection

My latest analysis centers around the 'Connect Button' issue. I think the issue is with how the event listener on `connect-server-btn` is handled in the shared navigation. The check during initial execution seems to be the main culprit. A safer approach is to simply run the relevant

code, but only if the dashboard element exists. I'm focusing now on integrating a `window.initDashboardMonitor` function and running it immediately, making it the most reliable solution.

## Enhancing Connection Control

I'm now integrating a `window.startDashboardMonitoring()` function, which will be accessible from the navigation bar. This approach directly addresses the "Connect" button's behavior, ensuring the dashboard correctly reflects the connection status after a successful connection event. I'm focusing on the interaction between `home_partial.html` and `navigation.html`, guaranteeing a reliable flow across the application, even after connections and partial page reloads. The goal is to make sure the dashboard is always updated and responsive.

## Resolving Execution Timing

My recent investigation centers around the timing of script execution. I've pinpointed the issue: `DOMContentLoaded` isn't firing on partial loads using HTMX. I will remove `DOMContentLoaded` and implement a named function for immediate execution, and added `window.startDashboardMonitoring()`. I am now confident that I can address the timing issue, which is the main blocking issue. Finally, I will modify `navigation.html` to call the new function.

## Triggering Dashboard Refresh

I'm now zeroing in on a direct solution. Given the user's "No Active Connection" issue, the core problem is the script failing to update the dashboard display. My proposed code directly executes the `startMonitoring()` function if `connectedServer` is found in `localStorage`, circumventing any `DOMContentLoaded` dependency. This should ensure the dashboard accurately reflects the connection status immediately after the connection success, fixing the displayed "No Active Connection". I'm making sure to test this change on both full and partial page loads to confirm the code will always refresh the dashboard, which should resolve the problem.

## Addressing Missing Data Retrieval

I'm now fully focused on the missing stats. I realize the user's `home.html` uses Django templating, which means `DOMContentLoaded` should fire on full page loads. However, looking at the logs, it's clear the `api/stats` endpoint isn't being requested when using HTMX or partial loads. I need to make the script handle both full and partial loads correctly. The primary challenge is ensuring the `startMonitoring()` function is called immediately if the page is loaded dynamically. To ensure this, I plan to update the Javascript to handle both page load scenarios. I will also make sure the `navigation.html` file calls the new `startMonitoring()` upon connection.

Vấn đề nằm ở chỗ: **Log của bạn KHÔNG HỀ CÓ dòng gọi API /remote/api/stats/**. Điều này chứng tỏ đoạn Script trong `home_partial.html` chưa bao giờ chạy, hoặc chạy nhưng bị dừng ngay lập tức.

Lý do phổ biến nhất là do cách bạn chuyển trang (nếu dùng HTMX hoặc Ajax để load nội dung vào thẻ `div`): sự kiện `DOMContentLoaded` chỉ chạy **1 lần duy nhất** khi tải lại cả trang web (F5). Khi bạn bấm chuyển qua lại

giữa các tab (Home, Process...), sự kiện này không chạy lại nữa, nên code JS trong `home_partial` bị "chết".

Dưới đây là cách sửa triệt để:

### 1. Sửa `home_partial.html` (Quan trọng nhất)

Thay vì dùng `addEventListener`, ta sẽ viết code để nó tự chạy ngay khi được load vào DOM, đồng thời gán hàm vào biến toàn cục `window` để `navigation` có thể gọi nó.

```
<script>
    // Dùng IIFE (Immediately Invoked Function Expression) để chạy ngay lập tức
    // nhưng vẫn giữ scope an toàn
    (function() {
        const dashboard = document.getElementById('live-dashboard');
        const placeholder = document.getElementById('monitor-placeholder');

        // Biến interval lưu vào window để có thể clear từ bên ngoài nếu cần
        window.dashboardInterval = null;

        // Hàm fetch dữ liệu
        async function fetchStats() {
            try {
                // Kiểm tra element còn tồn tại trên DOM không (phòng trường hợp
                // đã chuyển trang)
                if (!document.getElementById('live-dashboard')) {
                    stopMonitoring();
                    return;
                }

                const response = await fetch('/remote/api/stats/');
                if (!response.ok) throw new Error('Network error');

                const data = await response.json();

                if (data.status === 'disconnected') {
                    // Nếu Server báo disconnected -> Dừng monitor và gọi hàm dọn
                    // dẹp UI
                    stopMonitoring();
                    if (window.handleClientDisconnect)
                        window.handleClientDisconnect();
                    return;
                }

                if (data.status === 'success' && data.data) {
                    showDashboard(data.data);
                }
            } catch (error) {
                // console.log("Monitoring error/paused");
            }
        }
    })
```

```
function showDashboard(data) {
    if (!dashboard || !placeholder) return;

    dashboard.classList.remove('hidden');
    placeholder.classList.add('hidden');

    // Đień dữ liệu an toàn (kiểm tra element trước khi gán)
    const setTxt = (id, val) => { const el = document.getElementById(id);
if(el) el.textContent = val; };

    setTxt('dash-hostname', data.hostname);
    setTxt('dash-ip', data.internal_ip);
    setTxt('dash-os', data.os_info);
    setTxt('dash-latency', data.latency);
    setTxt('dash-ram', data.ram_free);
    setTxt('dash-battery', data.battery);

    // Update CPU
    const cpuVal = parseFloat(data.cpu_load);
    setTxt('dash-cpu', cpuVal);
    const bar = document.getElementById('dash-cpu-bar');
    if (bar) {
        bar.style.width = `${Math.min(cpuVal, 100)}%`;
        if(cpuVal > 80) bar.className = "bg-red-600 h-2.5 rounded-full transition-all duration-500";
        else if(cpuVal > 50) bar.className = "bg-yellow-500 h-2.5 rounded-full transition-all duration-500";
        else bar.className = "bg-blue-600 h-2.5 rounded-full transition-all duration-500";
    }
}

// Update Time
setTxt('dash-last-update', new Date().toLocaleTimeString());
}

// Gán hàm start/stop vào window để Navigation có thể gọi
window.startDashboardMonitoring = function() {
    console.log("⌚ Starting Dashboard Monitor...");
    fetchStats(); // Gọi ngay 1 lần

    // Xóa interval cũ nếu có để tránh trùng lặp
    if (window.dashboardInterval) clearInterval(window.dashboardInterval);

    window.dashboardInterval = setInterval(fetchStats, 2000);
};

function stopMonitoring() {
    if (dashboard) dashboard.classList.add('hidden');
    if (placeholder) placeholder.classList.remove('hidden');
    if (window.dashboardInterval) {
        clearInterval(window.dashboardInterval);
        window.dashboardInterval = null;
    }
}
```

```
// --- LOGIC CHẠY TỰ ĐỘNG ---
// Không dùng DOMContentLoaded nữa, chạy ngay lập tức khi script được load
const connectedServer = localStorage.getItem('connectedServer');
if (connectedServer) {
    window.startDashboardMonitoring();
} else {
    stopMonitoring();
}

})();
</script>
```

## 2. Sửa `navigation.html` (Để kích hoạt ngay khi bấm nút Connect)

Nếu người dùng đang ở trang Home mà bấm Connect, trang web không reload lại, nên ta cần gọi hàm `window.startDashboardMonitoring()` (mà ta vừa viết ở trên) ngay trong sự kiện click.

Tìm đến đoạn xử lý sự kiện `connectBtn` trong `navigation.html` và thêm đoạn code này vào lúc kết nối thành công:

```
// Trong navigation.html

// ... bên trong connectBtn.addEventListener ...
if (data.success) {
    showNotification('success', `Connected to ${selectedIp}`);

    // ... (code lưu localStorage cũ) ...
    saveConnectionState(serverInfo);

    // ... (code đổi nút connect/disconnect cũ) ...

    // === THÊM ĐOẠN NÀY ===
    // Nếu đang ở trang Home (có hàm startDashboardMonitoring), hãy chạy nó ngay!
    if (typeof window.startDashboardMonitoring === 'function') {
        window.startDashboardMonitoring();
    }
    // =====
}
```

## 3. Kiểm tra lại `server.cs`

Bạn nói đã làm "tất cả", nhưng hãy đảm bảo bạn đã:

1. Rebuild (Build lại) project C#.
2. Tắt Server cũ đang chạy.
3. Copy file `.exe` mới build ra chạy lại. (Vì nếu Server vẫn chạy bản cũ thì nó sẽ không hiểu lệnh `SYSTEM_INFO` và trả về lỗi hoặc không trả lời, làm script Python bị timeout).

Tóm tắt logic hoạt động sau khi sửa:

1. **Vào trang Home:** Script `home_partial` chạy ngay lập tức (không chờ DOMContentLoaded). Nó check `localStorage`. Nếu thấy đã connect -> Gọi `startDashboardMonitoring`.
2. **Bấm nút Connect:** `navigation.html` xử lý connect thành công -> Gọi `window.startDashboardMonitoring()` -> Dashboard hiện lên ngay lập tức mà không cần F5.

Prompt:

shell\_partial - HTML

Khoan, quay xe, trước khi bạn sửa html thì để tôi gửi bạn 1 file html mẫu, ví dụ như file remote shell CMD prompt mà tôi đã làm

Response:

Thinking:

Thinking:

### Envisioning the Dashboard

I'm now focusing on the server dashboard's core components: CPU load and RAM usage display. I'm exploring potential visualization methods for these metrics within the existing UI framework of the Remote Administration Tool. Thinking through the most efficient data retrieval and presentation methods is a priority right now, considering real-time updates.

### Expanding the Dashboard's Scope

I'm expanding the dashboard to include network latency, battery status, hostname, internal IP, and OS information upon connection. I've taken the user's "Khoan, quay xe" and `shell_partial.html` as a guide. The primary focus now shifts to mirroring the update approach used in the shell, specifically using HTMX for real-time adjustments and state management based on connection. I'm actively analyzing `initShellPage`, the polling mechanism, and the UI changes driven by connection states.

### Refactoring UI Logic

I'm now fully immersed in the refactoring process, focusing on `home_partial.html`. The goal is to align its JavaScript with the robust HTMX-driven pattern established in `shell_partial.html`. I'm meticulously dissecting the existing UI logic, particularly the `initShellPage` function, event listeners, and polling mechanisms. The aim is to create functions akin to `setUIConnected` and `setUIDisconnected` for my new data, ensuring seamless real-time updates and responsive UI elements tied to the connection state.

### Constructing API Endpoints

Now, I'm focusing on the Django backend implementation. I'm actively crafting the `/api/stats/` endpoint in `views.py` and configuring the associated URL in `urls.py`. The plan is to ensure this endpoint can efficiently receive and parse system statistics from the Python client.

My aim is to make the data accessible in a structured, parseable format from the Python client and usable by the front-end. After this, I'll go back to the [home\\_partial.html](#) file.

## Building the Polling Mechanism

I'm now concentrating on implementing the polling mechanism for the dashboard data. My current focus is crafting the [initDashboard](#) function and integrating it with the existing HTMX infrastructure. I'm exploring the most effective way to fetch data from the [/api/stats/](#) endpoint every two seconds and display it in the user interface. I need to ensure the dashboard initializes correctly after HTMX swaps elements. I am also working on [DashboardUI\(data\)](#) and [showDisconnectedState\(\)](#) functions, and attaching them to [htmx:afterSwap](#).

Đã hiểu ý bạn. Bạn muốn cấu trúc Javascript của trang Dashboard (Home) phải **đồng bộ** với cách viết trong [shell\\_partial.html](#):

1. Dùng hàm khởi tạo [init...](#) để tránh lỗi scope.
2. Gắn sự kiện [htmx:afterSwap](#) để chạy đúng khi chuyển trang.
3. Sử dụng cơ chế **Polling (hỏi liên tục)** để tự động cập nhật dữ liệu và phát hiện kết nối/ngắt kết nối mà không cần reload trang.

Dưới đây là bộ code hoàn chỉnh, tôi đã viết lại [home\\_partial.html](#) dựa trên mẫu [shell\\_partial.html](#) bạn cung cấp.

### 1. [server.cs](#) (Thêm tính năng lấy thông tin máy)

Bạn cần thêm thư viện [System.Diagnostics](#) (để dùng PerformanceCounter) và [System.Windows.Forms](#) (để lấy pin).

```
// Trong file server.cs

// 1. Thêm biến toàn cục ở đầu class server
PerformanceCounter cpuCounter;
PerformanceCounter ramCounter;

// 2. Trong Constructor public server()
public server()
{
    InitializeComponent();
    CheckForIllegalCrossThreadCalls = false;
    this.FormClosing += new FormClosingEventHandler(server_FormClosing);

    // --- KHỞI TẠO COUNTER ---
    try {
        // Lưu ý: CPU counter cần gọi NextValue() 2 lần hoặc đợi 1 nhịp mới có số chính xác,
        // nhưng ta gọi liên tục từ web nên nó sẽ tự chính xác dần.
        cpuCounter = new PerformanceCounter("Processor", "% Processor Time",
        "_Total");
        ramCounter = new PerformanceCounter("Memory", "Available MBytes");
    } catch { }
}
```

```
// 3. Trong HandleClientCommunication, switch(s)
switch (s)
{
    // ... các case cũ ...
    case "SYSTEM_INFO":
        send_system_info();
        break;
}

// 4. Thêm hàm xử lý ở cuối class
public void send_system_info()
{
    try
    {
        // Lấy CPU
        float cpu = 0;
        if (cpuCounter != null) cpu = cpuCounter.NextValue();

        // Lấy RAM Free
        float ram = 0;
        if (ramCounter != null) ram = ramCounter.NextValue();

        // Lấy Pin
        PowerStatus pwr = SystemInformation.PowerStatus;
        String battery = (pwr.BatteryLifePercent * 100).ToString() + "%";
        if (pwr.PowerLineStatus == PowerLineStatus.Online) battery += "
(Plugged)";

        // Lấy Hostname & OS
        String hostname = Dns.GetHostName();
        String os = Environment.OSVersion.ToString(); // VD: Microsoft Windows NT
10.0...

        // Lấy IP LAN (IPv4)
        String ipAddr = "Unknown";
        try {
            var host = Dns.GetHostEntry(Dns.GetHostName());
            foreach (var ip in host.AddressList) {
                if (ip.AddressFamily == AddressFamily.InterNetwork) {
                    ipAddr = ip.ToString();
                    break;
                }
            }
        } catch {}

        // Format gửi về: CPU|RAM|BATTERY|HOSTNAME|OS|IP
        // Ví dụ: 15.5|4096|85% (Plugged)|DESKTOP-ABC|Win 10|192.168.1.5
        String response = $"{cpu:0.0}|{ram}|{battery}|{hostname}|{os}|{ipAddr}";

        Program.nw.WriteLine(response);
        Program.nw.Flush();
    }
    catch (Exception ex)
```

```
{  
    Program.nw.WriteLine("ERROR|" + ex.Message);  
    Program.nw.Flush();  
}  
}
```

## 2. socket\_client\_persistent.py (Xử lý lệnh Python)

```
# Thêm vào class PersistentRemoteClient  
  
def get_system_stats(self):  
    """Lấy thông số hệ thống và tính Ping"""  
    if not self.connected:  
        return {"status": "error", "message": "Not connected"}  
  
    with self._lock:  
        try:  
            start_time = time.time() # Bấm giờ  
  
            self._send_str("SYSTEM_INFO")  
            response = self._recv_line()  
  
            end_time = time.time() # Ngắt giờ  
  
            # Tính Ping (ms)  
            latency = round((end_time - start_time) * 1000, 0) # Làm tròn  
  
            if response.startswith("ERROR"):  
                return {"status": "error", "message": response}  
  
            # Parse dữ liệu: CPU|RAM|BATTERY|HOSTNAME|OS|IP  
            parts = response.split('|')  
  
            # An toàn dữ liệu (tránh crash nếu thiếu field)  
            def get_part(idx, default="?"):  
                return parts[idx] if len(parts) > idx else default  
  
            data = {  
                "cpu_load": get_part(0, "0"),  
                "ram_free": get_part(1, "0"),  
                "battery": get_part(2, "Unknown"),  
                "hostname": get_part(3, "Unknown"),  
                "os_info": get_part(4, "Unknown"),  
                "internal_ip": get_part(5, "Unknown"),  
                "latency": latency  
            }  
  
            self._send_str("QUIT")  
            return {"status": "success", "data": data}  
  
        except Exception as e:
```

```
        self.connected = False # Đánh dấu ngắt kết nối nếu lỗi socket
        return {"status": "error", "message": str(e)}
```

### 3. `views.py` (API Endpoint)

```
# Thêm vào views.py

@require_http_methods(["GET"])
def get_server_stats(request):
    """
    API: Lấy thông số Dashboard
    Frontend gọi liên tục 2s/lần
    """
    client = _get_client(request)

    # 1. Nếu chưa có kết nối -> Trả về disconnected để frontend ẩn bảng
    if not client or not client.connected:
        return JsonResponse({
            "status": "disconnected",
            "message": "Client not connected"
        })

    # 2. Gửi lệnh lấy info
    result = client.get_system_stats()

    # 3. Nếu socket lỗi giữa chừng -> cũng báo disconnected
    if result.get("status") == "error" and "Broken pipe" in
        str(result.get("message")):
        return JsonResponse({"status": "disconnected"})

    return JsonResponse(result)
```

### 4. `urls.py`

```
# Thêm dòng này
path('api/stats/', views.get_server_stats, name='api_server_stats'),
```

### 5. `home_partial.html` (Giao diện chuẩn mẫu Shell)

Tôi đã viết lại hoàn toàn file này theo style của `shell_partial.html`: dùng `initHomeDashboard`, polling liên tục, và tự động ẩn/hiện dựa trên kết nối.

```
<div data-page="home-dashboard">
    <div id="dash-disconnected" class="text-center p-12 bg-gray-50 border border-
dashed border-gray-300 rounded-lg dark:bg-gray-800 dark:border-gray-700">
        <div class="inline-flex items-center justify-center w-16 h-16 rounded-full">
```

```
bg-gray-100 dark:bg-gray-700 mb-4">
    <svg class="w-8 h-8 text-gray-400" fill="none" stroke="currentColor"
viewBox="0 0 24 24"><path stroke-linecap="round" stroke-linejoin="round" stroke-
width="2" d="M12 6v6m0 0v6m0-6h6m-6 0H6"></path></svg>
</div>
<h3 class="text-lg font-medium text-gray-900 dark:text-white">No Active
Connection</h3>
<p class="text-gray-500 dark:text-gray-400 mt-2">Connect to a server to
view system performance.</p>
</div>

<div id="dash-connected" class="hidden transition-opacity duration-300">

    <div class="mb-4 bg-white border border-gray-200 rounded-lg shadow-sm
dark:border-gray-700 dark:bg-gray-800 p-4">
        <div class="flex flex-wrap justify-between items-center">
            <div>
                <h2 class="text-2xl font-bold text-gray-900 dark:text-white
flex items-center gap-3">
                    <span class="relative flex h-3 w-3">
                        <span class="animate-ping absolute inline-flex h-full w-
full rounded-full bg-green-400 opacity-75"></span>
                        <span class="relative inline-flex rounded-full h-3 w-3
bg-green-500"></span>
                    </span>
                    <span id="dash-hostname">Loading...</span>
                </h2>
                <div class="flex items-center gap-4 mt-2 text-sm text-gray-500
dark:text-gray-400">
                    <span class="flex items-center gap-1">
                        <svg class="w-4 h-4" fill="none" stroke="currentColor"
viewBox="0 0 24 24"><path stroke-linecap="round" stroke-linejoin="round" stroke-
width="2" d="M9.75 17L9 20l-1 1h8l-1-1-.75-3M3 13h18M5 17h14a2 2 0 002-2V5a2 2 0
00-2-2H5a2 2 0 00-2 2v10a2 2 0 002 2z"></path></svg>
                        <span id="dash-os">Checking OS...</span>
                    </span>
                    <span class="flex items-center gap-1">
                        <svg class="w-4 h-4" fill="none" stroke="currentColor"
viewBox="0 0 24 24"><path stroke-linecap="round" stroke-linejoin="round" stroke-
width="2" d="M21 12a9 9 0 01-9 9m9-9a9 9 0 00-9-9m9 9H3m9 9a9 9 0 01-9-9m9 9c1.657
0 3-4.03 3-9s-1.343-9-3-9m0 18c-1.657 0-3-4.03-3-9s1.343-9 3-9m-9 9a9 9 0 019-9">
</path></svg>
                        <span id="dash-ip">Checking IP...</span>
                    </span>
                </div>
            </div>
            <div class="text-right">
                <div class="text-3xl font-bold text-green-500"><span id="dash-
latency">--</span> <span class="text-sm font-normal text-gray-500">ms</span></div>
                <div class="text-xs text-gray-400">Network Latency</div>
            </div>
        </div>
    </div>
</div>
```

```
<div class="grid grid-cols-1 md:grid-cols-3 gap-4">

    <div class="p-4 bg-white border border-gray-200 rounded-lg shadow-sm dark:bg-gray-800 dark:border-gray-700">
        <div class="flex justify-between mb-2">
            <span class="text-sm font-medium text-gray-500 dark:text-gray-400">CPU Load</span>
                <svg class="w-5 h-5 text-blue-500" fill="none" stroke="currentColor" viewBox="0 0 24 24"><path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M9 3v2m6-2v2M9 19v2m6-2v2M5 9H3m2 6H3m18-6h-2m2 6h-2M7 19h10a2 2 0 002-2V7a2 2 0 00-2-2H7a2 2 0 00-2 2v10a2 2 0 002 2zM9 9h6v6H9V9z"></path></svg>
        </div>
        <div class="flex items-baseline gap-2">
            <span id="dash-cpu" class="text-3xl font-bold text-gray-900 dark:text-white">0</span>
            <span class="text-sm text-gray-500">%</span>
        </div>
        <div class="w-full bg-gray-200 rounded-full h-2.5 dark:bg-gray-700 mt-3">
            <div id="dash-cpu-bar" class="bg-blue-600 h-2.5 rounded-full transition-all duration-500" style="width: 0%"></div>
        </div>
    </div>

    <div class="p-4 bg-white border border-gray-200 rounded-lg shadow-sm dark:bg-gray-800 dark:border-gray-700">
        <div class="flex justify-between mb-2">
            <span class="text-sm font-medium text-gray-500 dark:text-gray-400">RAM Available</span>
                <svg class="w-5 h-5 text-purple-500" fill="none" stroke="currentColor" viewBox="0 0 24 24"><path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M19 11H5m14 0a2 2 0 012 2v6a2 2 0 01-2 2H5a2 2 0 01-2-2v-6a2 2 0 012-2m14 0V9a2 2 0 00-2-2M5 11V9a2 2 0 012-2m0 0V5a2 2 0 012-2h6a2 2 0 012 2v2M7 7h10"></path></svg>
        </div>
        <div class="flex items-baseline gap-2">
            <span id="dash-ram" class="text-3xl font-bold text-gray-900 dark:text-white">0</span>
            <span class="text-sm text-gray-500">MB</span>
        </div>
        <p class="text-xs text-gray-500 mt-3">Free memory</p>
    </div>

    <div class="p-4 bg-white border border-gray-200 rounded-lg shadow-sm dark:bg-gray-800 dark:border-gray-700">
        <div class="flex justify-between mb-2">
            <span class="text-sm font-medium text-gray-500 dark:text-gray-400">Battery</span>
                <svg class="w-5 h-5 text-yellow-500" fill="none" stroke="currentColor" viewBox="0 0 24 24"><path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M13 10V3L4 14h7v7l9-11h-7z"></path></svg>
        </div>
        <div class="flex items-baseline gap-2">
```

```
        <span id="dash-battery" class="text-2xl font-bold text-gray-900 dark:text-white">--</span>
    </div>
    <p class="text-xs text-gray-500 mt-3">Power Status</p>
</div>
</div>
</div>
</div>

<script>
function initHomeDashboard() {
    // Chỉ chạy nếu đang ở trang home-dashboard
    if (!document.querySelector('[data-page="home-dashboard"]')) return;

    const viewDisconnected = document.getElementById('dash-disconnected');
    const viewConnected = document.getElementById('dash-connected');

    let updateInterval = null;
    let isConnectedState = false;

    // --- HÀM CẬP NHẬT UI ---
    function setUIState(connected) {
        // Chỉ update DOM khi trạng thái thực sự thay đổi để tránh repaint
        if (connected === isConnectedState) return;
        isConnectedState = connected;

        if (connected) {
            viewDisconnected.classList.add('hidden');
            viewConnected.classList.remove('hidden');
        } else {
            viewConnected.classList.add('hidden');
            viewDisconnected.classList.remove('hidden');
        }
    }

    function updateData(data) {
        // Helper điền text
        const setText = (id, val) => {
            const el = document.getElementById(id);
            if(el) el.textContent = val;
        };

        setText('dash-hostname', data.hostname);
        setText('dash-os', data.os_info);
        setText('dash-ip', data.internal_ip);
        setText('dash-latency', data.latency);
        setText('dash-ram', data.ram_free);
        setText('dash-battery', data.battery);

        // Xử lý thanh CPU
        const cpuVal = parseFloat(data.cpu_load);
        setText('dash-cpu', cpuVal);

        const bar = document.getElementById('dash-cpu-bar');
    }
}
```

```
if (bar) {
    bar.style.width = `${Math.min(cpuVal, 100)}%`;

    // Đổi màu theo mức độ load
    let colorClass = "bg-blue-600";
    if(cpuVal > 80) colorClass = "bg-red-600";
    else if(cpuVal > 50) colorClass = "bg-yellow-500";

    bar.className = `${colorClass} h-2.5 rounded-full transition-all duration-500`;
}

// --- HÀM POLLING CHÍNH ---
async function fetchStats() {
    // Kiểm tra nếu đã chuyển trang khác thì dừng ngay
    if (!document.querySelector('[data-page="home-dashboard"]')) {
        if(updateInterval) clearInterval(updateInterval);
        return;
    }

    try {
        const response = await fetch('/remote/api/stats/');
        // Nếu 404 hoặc lỗi mạng -> coi như disconnect
        if (!response.ok) throw new Error("Network error");

        const data = await response.json();

        if (data.status === 'success') {
            setUIState(true); // Đã kết nối
            updateData(data.data);
        } else {
            // status: 'disconnected' hoặc 'error'
            setUIState(false);
        }
    } catch (error) {
        // Lỗi mạng hoặc server sập
        setUIState(false);
    }
}

// --- KHỞI ĐỘNG LOOP ---
// Gọi ngay lần đầu
fetchStats();

// Loop mỗi 2 giây (tương tự polling CMD nhưng nhanh hơn chút vì cần realtime)
updateInterval = setInterval(fetchStats, 2000);
}

// Đăng ký sự kiện giống shell_partial để chạy được với HTMX
document.addEventListener('htmx:afterSwap', initHomeDashboard);
```

```
document.addEventListener('DOMContentLoaded', initHomeDashboard);  
</script>
```

Cách hoạt động (Giống hệ Remote CMD):

1. **Server C#**: Thêm khả năng đọc CPU/RAM.
2. **Home Page**: Khi load vào, hàm `initHomeDashboard` chạy.
3. **Polling**: Javascript gửi request `/api/stats/` mỗi 2 giây.
4. **Auto-Detect**:
  - Nếu trả về data thành công -> Hiện Dashboard, ẩn "No Active Connection".
  - Nếu trả về `disconnected` hoặc lỗi -> Ẩn Dashboard, hiện "No Active Connection".
5. **Clean**: Tự động kiểm tra `data-page` để dừng polling khi bạn chuyển sang trang khác (Processes, Application...).

**Prompt:**

Tôi không biết có phải tại tính năng mới hay không, nhưng mà bây giờ web chạy rất chậm.

Cái server bên C# cũng cần rất lâu để start up, không hiểu tại sao. Đến cả connect vào server thôi bây giờ nó cũng chậm như rùa, mặc dù đang làm trên cùng máy...

```
[16/Dec/2025 14:40:41] "GET /remote/processes/ HTTP/1.1" 200 14139
```

```
[16/Dec/2025 14:40:42] "GET /remote/home/ HTTP/1.1" 200 9881
```

```
[16/Dec/2025 14:40:44] "GET /remote/applications/ HTTP/1.1" 200 14421
```

```
[16/Dec/2025 14:40:44] "GET /remote/home/ HTTP/1.1" 200 9881
```

```
[16/Dec/2025 14:41:35] "GET /remote/api/stats/ HTTP/1.1" 200 177
```

Vả lại trên trang web, khi ấn vào cái tab lần đầu thì chạy được, nhưng những lần tiếp theo (sau khi switch tab) thì nó hết chạy.

Ý tôi không phải muốn bạn làm Y CHANG bên CMD, tôi chỉ gửi bạn 1 cái mẫu để bạn làm reference thôi.

**Response:**

Thinking:

Thinking:

### Assessing the Bottlenecks

I'm deep into analyzing the user's performance issues. Specifically, I've identified the web application's sluggishness and the server's glacial startup as key symptoms. Now, I'm trying to pinpoint where the bottlenecks lie. Initial investigations suggest these issues stem from resource constraints. I am leaning into exploring solutions relating to resource optimization.

### Identifying Performance Counter Issues

I'm now investigating performance counter initialization and its impact. The connection is slow as molasses, even locally, hinting at a systemic issue. The dashboard's "System Info" section is unreliable, ceasing updates after tab navigation. This suggests potential errors with performance counter interactions. There's a notable delay between page load and the first API call, which hints at either a frontend polling issue or the server hanging. I must prioritize fixing the tab switching logic.

### **Addressing Initialization Latency**

I'm now zeroing in on the performance counter initialization. It's a known C# issue and seems to be the root of both the slow startup and sluggish connection. I've realized initializing in the server constructor blocks the UI. Also, the first `NextValue()` call blocks, so frequent polling is inefficient. I need to move the counter setup to a background task and implement an asynchronous mechanism for fetching values.

### **Analyzing UI Re-Initialization**

I'm now focusing on a tricky UI refresh. Previously, after a swap, the existing code wasn't correctly re-initializing. This likely stems from lingering intervals or the incorrect event listener for the HTMX swap. It's key to make sure `initHomeDashboard` gets called after a swap. Potential event listener issues or ID checks are the likely culprits for missed invocations.

### **Improving Metric Gathering**

I've realized the problem lies with the current C# code's `NextValue()` calls occurring within the socket handling loop, leading to significant delays. The timer-based background update of a global variable seems like a good path. To recap, a `Timer` in C# will update stats asynchronously. The socket command will then read instantly from that variable. I also see that caching performance counter values makes sense to avoid recreation. Finally, I'm considering alternatives to `PerformanceCounter`, particularly for CPU metrics.

### **Streamlining JS Re-initialization**

I'm now revising the JavaScript to address the re-initialization failure after tab switches. I've pinpointed the issues: potential scope pollution from `window.dashboardInterval` and the need for more robust cleanup. I'm focusing on ensuring a clean re-initialization of the polling mechanism after each HTMX swap. To that end, the implementation now includes a check for element existence, clearing existing intervals before creating new ones, and targeting the specific content area for `htmx:afterSwap` to guarantee proper event handling. I am also working on `htmx:historyRestore` handling to ensure updates on back navigations.

### **Finalizing Code Adjustments**

I've just finished the code modifications, implementing the timer-based background update in C# to resolve the thread-blocking issue. The socket now retrieves values from the cached information, resulting in quicker retrieval. On the JavaScript side, I have ensured proper cleanup before swapping and correct re-initialization after tab switches, addressing the re-initialization failure with a check for element existence. I also confirmed all intervals were cleared and proper event handling takes place. The solution will involve a check for `null` values as well. I hope this addresses the user's issue with slowness.

## Revising for HTMX Integration

I'm now revising my approach. I realized I must account for HTMX's behavior more precisely. Previously, the approach was to use HTMX swaps which caused issues. The focus is to make the partials self-contained and self-initializing. I also need to make sure the server-side metrics gathering code is decoupled from the socket. I've re-written the metric gathering to be background.

Vấn đề bạn gặp phải rất phổ biến khi làm việc với `PerformanceCounter` trong C#.

**Nguyên nhân gây chậm (Lag):** Hàm `cpuCounter.NextValue()` hoạt động bằng cách lấy mẫu (sampling) hiệu năng hệ thống. Để tính toán % CPU, nó cần so sánh giữa 2 thời điểm. Khi bạn gọi hàm này trực tiếp trong luồng xử lý Socket (lệnh `SYSTEM_INFO`), **toàn bộ Server sẽ bị "đơ" lại (block)** trong khoảng vài trăm mili-giây đến 1 giây để chờ Windows trả về số liệu. Điều này khiến kết nối mạng bị chậm như rùa vì Server bận tính toán thay vì phản hồi Client.

### Giải pháp:

- Server C#:** Không tính toán CPU khi có lệnh gửi đến. Thay vào đó, dùng một **Timer chạy ngầm** (Background Timer) cứ 1 giây cập nhật số liệu vào biến toàn cục một lần. Khi Client hỏi, Server chỉ việc trả về biến đó ngay lập tức (0ms delay).
- Frontend JS:** Sửa lại logic khởi tạo để đảm bảo khi switch tab (HTMX swap), code JS được kích hoạt lại đúng cách.

Dưới đây là cách sửa:

#### 1. Sửa `server.cs` (Quan trọng nhất - Giải quyết Lag)

Bạn thay đổi cấu trúc của class `server` như sau: tách việc lấy chỉ số ra khỏi việc gửi dữ liệu.

```
// ... các using khác ...
using System.Timers; // Thêm dòng này để dùng Timer xịn hơn

namespace ServerApp
{
    public partial class server : Form
    {
        // --- 1. BIẾN TOÀN CỤC ---
        Thread serverThread;
        Thread udpDiscoveryThread;
        Thread tklog = null;

        // Timer để cập nhật thông số hệ thống ngầm
        System.Timers.Timer statsTimer;

        // Cache lưu trữ dữ liệu hệ thống (để gửi ngay lập tức)
        string cachedSystemInfo = "0|0|Checking...|...|...|...";

        PerformanceCounter cpuCounter;
        PerformanceCounter ramCounter;
```

```
// ... Webcams, ScreenRecorders instances ...

public server()
{
    InitializeComponent();
    CheckForIllegalCrossThreadCalls = false;
    this.FormClosing += new FormClosingEventHandler(server_FormClosing);

    // --- 2. KHỞI TẠO TIMER & COUNTER ---
    try {
        // Khởi tạo Counter (vẫn có thể gây delay nhẹ lúc bật app, nhưng
        chỉ 1 lần)
        cpuCounter = new PerformanceCounter("Processor", "% Processor
Time", "_Total");
        ramCounter = new PerformanceCounter("Memory", "Available MBytes");

        // Gọi lần đầu (thường trả về 0) để làm nóng Counter
        cpuCounter.NextValue();

        // Cấu hình Timer chạy mỗi 1000ms (1 giây)
        statsTimer = new System.Timers.Timer(1000);
        statsTimer.Elapsed += UpdateSystemStats; // Gán hàm xử lý
        statsTimer.AutoReset = true;
        statsTimer.Enabled = true; // Bắt đầu chạy ngay
    } catch { }

    // ... code xóa log cũ ...
}

// --- 3. HÀM CHẠY NGẦM (UpdateSystemStats) ---
// Hàm này chạy riêng biệt, không liên quan đến Socket nên không làm lag
mạng
private void UpdateSystemStats(Object source, ElapsedEventArgs e)
{
    try
    {
        float cpu = cpuCounter.NextValue();
        float ram = ramCounter.NextValue();

        // Lấy Pin (cần Invoke nếu lỗi thread, nhưng SystemInformation
        thường an toàn)
        String battery = "Unknown";
        try {
            PowerStatus pwr = SystemInformation.PowerStatus;
            battery = (pwr.BatteryLifePercent * 100).ToString() + "%";
            if (pwr.PowerLineStatus == PowerLineStatus.Online) battery +=
" (Plugged)";
        } catch {}

        String hostname = Dns.GetHostName();
        String os = Environment.OSVersion.ToString();

        String ipAddr = "Unknown";
        try {
```

```

        var host = Dns.GetHostEntry(Dns.GetHostName());
        foreach (var ip in host.AddressList) {
            if (ip.AddressFamily == AddressFamily.InterNetwork) {
                ipAddr = ip.ToString();
                break;
            }
        }
    } catch {}

    // Cập nhật vào biến cache
    cachedSystemInfo = $"{cpu:0.0}|{ram}|{battery}|{hostname}|{os}|{ipAddr}";
}
catch { }
}

// --- 4. SỬA HÀM GỬI INFO (send_system_info) ---
// Hàm này giờ chỉ việc đọc cache -> Cực nhanh
public void send_system_info()
{
    try
    {
        // Gửi ngay lập tức dữ liệu đã được Timer chuẩn bị sẵn
        Program.nw.WriteLine(cachedSystemInfo);
        Program.nw.Flush();
    }
    catch (Exception ex)
    {
        Program.nw.WriteLine("ERROR| " + ex.Message);
        Program.nw.Flush();
    }
}

// ... Các phần còn lại giữ nguyên ...

```

## 2. Sửa `home_partial.html` (Giải quyết lỗi mất tính năng khi switch tab)

Vấn đề là khi bạn dùng HTMX để chuyển tab, nội dung HTML cũ bị xóa đi, nhưng các Event Listener cũ hoặc Timer cũ có thể vẫn còn hoặc bị mất context.

Cách sửa tốt nhất cho HTMX: Dùng sự kiện `htmx:load` (nó kích hoạt cho cả lần tải trang đầu tiên VÀ mỗi khi nội dung được swap vào).

Bạn thay thế toàn bộ nội dung file `home_partial.html` bằng code này:

```

<div id="dashboard-root" data-page="home-dashboard">
    <div id="dash-disconnected" class="text-center p-12 bg-gray-50 border border-dashed border-gray-300 rounded-lg dark:bg-gray-800 dark:border-gray-700">
        <div class="inline-flex items-center justify-center w-16 h-16 rounded-full bg-gray-100 dark:bg-gray-700 mb-4">
            <svg class="w-8 h-8 text-gray-400" fill="none" stroke="currentColor"

```

```
viewBox="0 0 24 24"><path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M12 6v6m0 0v6m-6h6m-6 0H6"></path></svg>
</div>
<h3 class="text-lg font-medium text-gray-900 dark:text-white">No Active Connection</h3>
<p class="text-gray-500 dark:text-gray-400 mt-2">Connect to a server to view system performance.</p>
</div>

<div id="dash-connected" class="hidden transition-opacity duration-300">

    <div class="mb-4 bg-white border border-gray-200 rounded-lg shadow-sm dark:border-gray-700 dark:bg-gray-800 p-4">
        <div class="flex flex-wrap justify-between items-center">
            <div>
                <h2 class="text-2xl font-bold text-gray-900 dark:text-white flex items-center gap-3">
                    <span class="relative flex h-3 w-3">
                        <span class="animate-ping absolute inline-flex h-full w-full rounded-full bg-green-400 opacity-75"></span>
                        <span class="relative inline-flex rounded-full h-3 w-3 bg-green-500"></span>
                    </span>
                    <span id="dash-hostname">Loading...</span>
                </h2>
                <div class="flex items-center gap-4 mt-2 text-sm text-gray-500 dark:text-gray-400">
                    <span id="dash-os">Checking OS...</span> | <span id="dash-ip">Checking IP...</span>
                </div>
            </div>
            <div class="text-right">
                <div class="text-3xl font-bold text-green-500"><span id="dash-latency">---</span> <span class="text-sm font-normal text-gray-500">ms</span></div>
                <div class="text-xs text-gray-400">Latency</div>
            </div>
        </div>
    </div>
</div>

<div class="grid grid-cols-1 md:grid-cols-3 gap-4">
    <div class="p-4 bg-white border border-gray-200 rounded-lg shadow-sm dark:bg-gray-800 dark:border-gray-700">
        <div class="flex justify-between mb-2"><span class="text-gray-500 dark:text-gray-400">CPU Load</span></div>
        <div class="flex items-baseline gap-2">
            <span id="dash-cpu" class="text-3xl font-bold text-gray-900 dark:text-white">0</span><span class="text-sm">%</span>
        </div>
        <div class="w-full bg-gray-200 rounded-full h-2.5 dark:bg-gray-700 mt-3">
            <div id="dash-cpu-bar" class="bg-blue-600 h-2.5 rounded-full transition-all duration-500" style="width: 0%"></div>
        </div>
    </div>
</div>
```

```
<div class="p-4 bg-white border border-gray-200 rounded-lg shadow-sm dark:bg-gray-800 dark:border-gray-700">
    <div class="flex justify-between mb-2"><span class="text-gray-500 dark:text-gray-400">RAM Free</span></div>
    <div class="flex items-baseline gap-2">
        <span id="dash-ram" class="text-3xl font-bold text-gray-900 dark:text-white">0</span><span class="text-sm">MB</span>
    </div>
</div>
<div class="p-4 bg-white border border-gray-200 rounded-lg shadow-sm dark:bg-gray-800 dark:border-gray-700">
    <div class="flex justify-between mb-2"><span class="text-gray-500 dark:text-gray-400">Battery</span></div>
    <div class="flex items-baseline gap-2">
        <span id="dash-battery" class="text-2xl font-bold text-gray-900 dark:text-white">--</span>
    </div>
</div>
</div>

<script>
    // Logic được đóng gói trong hàm init
    function initDashboardFeature(elt) {
        // Chỉ chạy nếu element cha (dashboard-root) tồn tại
        const root = document.getElementById('dashboard-root');
        if (!root) return;

        console.log("⚡ Dashboard initialized");

        // Dọn dẹp interval cũ nếu có (tránh leak khi reload nhanh)
        if (window.dashInterval) clearInterval(window.dashInterval);

        const viewDisconnected = document.getElementById('dash-disconnected');
        const viewConnected = document.getElementById('dash-connected');

        function setUI(connected) {
            if(connected) {
                viewDisconnected.classList.add('hidden');
                viewConnected.classList.remove('hidden');
            } else {
                viewConnected.classList.add('hidden');
                viewDisconnected.classList.remove('hidden');
            }
        }

        async function updateStats() {
            // Kiểm tra: Nếu element đã bị xóa khỏi DOM (do chuyển tab), dừng loop
            if (!document.getElementById('dashboard-root')) {
                console.log("🔴 Dashboard removed from DOM. Stopping interval.");
                if (window.dashInterval) clearInterval(window.dashInterval);
                return;
            }
        }
    }
}
```

```
}

try {
    const response = await fetch('/remote/api/stats/');
    if (!response.ok) throw new Error("Net Error");
    const data = await response.json();

    if (data.status === 'success') {
        setUI(true);

        // Update DOM
        document.getElementById('dash-hostname').textContent =
data.data.hostname;
        document.getElementById('dash-os').textContent =
data.data.os_info;
        document.getElementById('dash-ip').textContent =
data.data.internal_ip;
        document.getElementById('dash-latency').textContent =
data.data.latency;
        document.getElementById('dash-ram').textContent =
data.data.ram_free;
        document.getElementById('dash-battery').textContent =
data.data.battery;

        const cpu = parseFloat(data.data.cpu_load);
        document.getElementById('dash-cpu').textContent = cpu;
        const bar = document.getElementById('dash-cpu-bar');
        if(bar) {
            bar.style.width = `${Math.min(cpu, 100)}%`;
            bar.className = (cpu > 80 ? "bg-red-600" : (cpu > 50 ?
"bg-yellow-500" : "bg-blue-600")) + " h-2.5 rounded-full transition-all duration-500";
        }
    } else {
        setUI(false); // Disconnected
    }
} catch (e) {
    setUI(false);
}
}

// Chạy ngay 1 lần
updateStats();
// Set interval mới
window.dashInterval = setInterval(updateStats, 2000);
}

// --- SỬ DỤNG htmx:load ---
// Sự kiện này chạy mỗi khi HTMX thêm content mới vào DOM
// Nó bao phủ cả trường hợp F5 và chuyển Tab
htmx.onLoad(function(elt) {
    // Chỉ init nếu nội dung vừa load có chứa dashboard-root
    if (elt.querySelector && elt.querySelector('#dashboard-root')) {
```

```

        initDashboardFeature(elt);
    } else if (elt.id === 'dashboard-root') {
        initDashboardFeature(elt);
    }
});
</script>

```

Tại sao sửa như thế này lại giải quyết được vấn đề?

- Server C#:** Việc chuyển tính toán CPU sang Timer chạy nền (1s/lần) giúp lệnh Socket phản hồi ngay lập tức. Kết nối sẽ nhanh trở lại.
- JS `htmx.onLoad`:** Thay vì dùng `DOMContentLoaded` (chỉ chạy 1 lần khi F5) hay `afterSwap` (có thể bị miss), `htmx.onLoad` là cách chuẩn nhất để kích hoạt Javascript cho các phần tử mới được thêm vào bởi HTMX.
- Check tồn tại:** Trong vòng lặp `setInterval`, tôi thêm dòng `if (!document.getElementById('dashboard-root')) ... clearInterval....` Điều này đảm bảo khi bạn chuyển sang tab Processes, vòng lặp của Dashboard sẽ tự chết đi, không chạy ngầm gây lỗi.

Prompt:

keylogger\_partial - HTML

screenshot\_partial - HTML

screen\_partial - HTML

Tôi nghĩ nguyên nhân là do cái tab của bạn á. Lúc mới mở web lên mọi thứ chạy rất bình thường, không lag không gì cả. Nhưng một khi bấm vào cái tab home xong là...mọi thứ rất rất chậm, ý tôi chậm ở đây là khoảng 30s web mới nhận được 1 lệnh á.

À với lại tình trạng thoát ra xong vô lại nó hiện chưa connect vẫn còn nha

Cái file .html của bạn thật sự nhìn rất khác mấy cái file .html khác. Để tôi gửi bạn vài file tham khảo:

Response:

Thinking:

Thinking:

### Identifying the Lag

I've pinpointed the issue: the web interface crawls to a halt after clicking the "Home" tab, with commands taking an agonizing 30 seconds to react. This lag is the central problem I need to address. The focus is now on the source of that slow-down, since it is very specifically located after a specific event. I'm focusing on that, and nothing else.

### Addressing Connection State

I've realized the UI often misrepresents the connection status, showing "Disconnected" even when the server might still be connected. I'm investigating how the UI incorrectly reports the

connection state. I also noticed that the `home_partial.html` structure deviates from other partials, so I am comparing that. The polling mechanism (`setInterval`) for `/remote/api/stats/` now seems to be central to this.

## Analyzing Server Blocking

I'm now zeroing in on server-side blocking as the likely bottleneck. The 30-second delay strongly suggests a timeout, prompting me to examine potential blocking calls between the Python and C# components. I suspect the Django view might be hanging due to synchronous or slow socket operations, especially with continuous polling from the "Home" tab. This aligns with the user's observation of the "Home" tab's impact on overall speed.

## Revising UI Robustness

I'm now focusing on a critical flaw: the UI's connection logic. Even when `localStorage` indicates connectivity, the initial `fetchStats` can fail due to server restarts or session issues, resulting in an immediate "disconnected" state. I'm prioritizing more resilient error handling within the UI. Specifically, I'm aiming to modify the code so that it continuously attempts to fetch data if `localStorage` reports a connected state, switching to disconnected only after *repeated* failures, instead of a single error. I'm also ensuring the `/remote/api/stats/` endpoint is properly robust.

## Streamlining the Dashboard

I'm now revising the `home_partial.html` structure. I'll align it with the `<main data-page="home">` structure for consistency and re-evaluate my handling of the `dashboard-root` ID. I'm focusing on the Python side, since even with C# timers, a slow or unstable connection can cause a queue of requests due to aggressive polling from the Dashboard. I plan to increase the interval and implement a shorter timeout for the Python socket calls, so that the web server doesn't hang.

## Improving Polling Robustness

I've implemented a check in the polling loop to prevent sending multiple requests if one is already pending. I'm also re-evaluating `views.py` to ensure `get_server_stats` is optimized. The C# server fix (Timer) is crucial. Furthermore, I've addressed the "No Active Connection" issue on refresh by updating the logic to only hide the dashboard if the API explicitly returns a disconnected status, rather than on any network error.

Tôi đã hiểu vấn đề. Việc web bị "treo" 30 giây khi bật tab Home là do **tắc nghẽn Request**.

Do server Django mặc định là đơn luồng (single-threaded) hoặc ít luồng, nếu Tab Home gửi yêu cầu lấy thông tin liên tục (Polling) mà Server C# trả lời chậm hoặc Python bị kẹt khi đọc Socket, thì **toàn bộ Web sẽ bị treo** chờ cái request đó xong mới xử lý tiếp cái khác (như chuyển tab).

Dưới đây là giải pháp đồng bộ phong cách code với các file bạn gửi (`data-page`, cấu trúc HTML/JS tách biệt) và xử lý triệt để vụ lag.

### 1. File `home_partial.html` (Viết lại hoàn toàn)

Tôi đã sửa lại theo đúng mẫu `keylogger_partial.html` của bạn:

1. Dùng thẻ `<main data-page="home">`.
2. **Cơ chế "Anti-Flood"**: Chỉ gửi request mới khi request cũ đã hoàn thành. Không dùng `setInterval` cứing nhắc nữa mà dùng `setTimeout` sau khi nhận phản hồi.
3. **Xử lý UI**: Kiểm tra `localStorage` ngay lập tức để hiện giao diện Dashboard, không chờ API trả về mới hiện (tránh cảm giác "chưa connect").

```
<main data-page="home">
  <div class="px-4 pt-6">

    <div id="dash-disconnected" class="hidden">
      <div class="p-8 text-center bg-white border border-gray-200 rounded-lg shadow-sm dark:bg-gray-800 dark:border-gray-700">
        <div class="inline-flex items-center justify-center w-16 h-16 mb-4 bg-gray-100 rounded-full dark:bg-gray-700">
          <svg class="w-8 h-8 text-gray-400" fill="none" stroke="currentColor" viewBox="0 0 24 24"><path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M12 6v6m0 0v6m0-6h6m-6 0H6"></path></svg>
        </div>
        <h3 class="text-lg font-medium text-gray-900 dark:text-white">No Active Connection</h3>
        <p class="mt-2 text-gray-500 dark:text-gray-400">Please select a server and click "Connect" to view dashboard.</p>
      </div>
    </div>

    <div id="dash-connected" class="hidden space-y-4">

      <div class="p-4 bg-white border border-gray-200 rounded-lg shadow-sm dark:border-gray-700 dark:bg-gray-800">
        <div class="flex flex-wrap items-center justify-between">
          <div>
            <h2 class="flex items-center gap-3 text-2xl font-bold text-gray-900 dark:text-white">
              <span class="relative flex w-3 h-3">
                <span class="absolute inline-flex w-full h-full bg-green-400 rounded-full opacity-75 animate-ping"></span>
                <span class="relative inline-flex w-3 h-3 bg-green-500 rounded-full"></span>
              </span>
              <span id="dash-hostname">Loading...</span>
            </h2>
            <div class="flex items-center gap-4 mt-2 text-sm text-gray-500 dark:text-gray-400">
              <span id="dash-os">Checking OS...</span> | <span id="dash-ip">Checking IP...</span>
            </div>
          </div>
          <div class="text-right">
            <div class="text-3xl font-bold text-green-500"><span id="dash-latency">---</span> <span class="text-sm font-normal text-gray-500">ms</span></div>
            <div class="text-xs text-gray-400">Latency</div>
          </div>
        </div>
      </div>
    </div>
  </div>
</main>
```

```
        </div>
    </div>
</div>

<div class="grid grid-cols-1 gap-4 md:grid-cols-3">

    <div class="p-4 bg-white border border-gray-200 rounded-lg shadow-sm dark:bg-gray-800 dark:border-gray-700">
        <div class="flex justify-between mb-2"><span class="text-sm font-medium text-gray-500 dark:text-gray-400">CPU Load</span></div>
        <div class="flex items-baseline gap-2">
            <span id="dash-cpu" class="text-3xl font-bold text-gray-900 dark:text-white">0</span><span class="text-sm">%</span>
        </div>
        <div class="w-full h-2.5 mt-3 bg-gray-200 rounded-full dark:bg-gray-700">
            <div id="dash-cpu-bar" class="h-2.5 bg-blue-600 rounded-full transition-all duration-500" style="width: 0%"></div>
        </div>
    </div>

    <div class="p-4 bg-white border border-gray-200 rounded-lg shadow-sm dark:bg-gray-800 dark:border-gray-700">
        <div class="flex justify-between mb-2"><span class="text-sm font-medium text-gray-500 dark:text-gray-400">RAM Free</span></div>
        <div class="flex items-baseline gap-2">
            <span id="dash-ram" class="text-3xl font-bold text-gray-900 dark:text-white">0</span><span class="text-sm">MB</span>
        </div>
        <p class="mt-2 text-xs text-gray-400">Available Memory</p>
    </div>

    <div class="p-4 bg-white border border-gray-200 rounded-lg shadow-sm dark:bg-gray-800 dark:border-gray-700">
        <div class="flex justify-between mb-2"><span class="text-sm font-medium text-gray-500 dark:text-gray-400">Battery</span></div>
        <div class="flex items-baseline gap-2">
            <span id="dash-battery" class="text-2xl font-bold text-gray-900 dark:text-white">--</span>
        </div>
        <p class="mt-2 text-xs text-gray-400">Power Status</p>
    </div>
</div>
</div>

<script>
function initHomePage() {
    // 1. Kiểm tra xem có đang ở trang Home không
    if (!document.querySelector('[data-page="home"]')) return;

    console.log("INIT HOME PAGE");
}
```

```
const viewDisconnected = document.getElementById('dash-disconnected');
const viewConnected = document.getElementById('dash-connected');
let pollingTimer = null;
let isPageActive = true; // Cờ kiểm soát khi chuyển trang

// 2. Hàm set UI (Ẩn/Hiện Dashboard)
function setUI(isConnected) {
    if (isConnected) {
        viewDisconnected.classList.add('hidden');
        viewConnected.classList.remove('hidden');
    } else {
        viewConnected.classList.add('hidden');
        viewDisconnected.classList.remove('hidden');
    }
}

// 3. Hàm fetch dữ liệu (Dùng đệ quy setTimeout thay vì setInterval để
// chống nghẽn)
async function fetchStats() {
    // Nếu đã chuyển trang -> Dừng ngay
    if (!document.querySelector('[data-page="home"]')) return;

    try {
        // Kiểm tra localStorage trước khi gọi API
        // Nếu localStorage nói chưa connect thì không gọi API làm gì cho
        // tốn tài nguyên
        if (!localStorage.getItem('connectedServer')) {
            setUI(false);
            return; // Dừng polling
        }

        const response = await fetch('/remote/api/stats/');
        // Nếu server Django đang bận hoặc lỗi mạng -> 404/500
        if (!response.ok) throw new Error("Network/Server Busy");

        const data = await response.json();

        if (data.status === 'success') {
            setUI(true);
            updateDashboardData(data.data);
        } else if (data.status === 'disconnected') {
            // Server xác nhận đã ngắt kết nối
            setUI(false);
            // Có thể gọi window.handleClientDisconnect() ở đây nếu muốn
            đồng bộ Navigation
        } else {
            // Lỗi khác nhưng vẫn coi là connected (vd: timeout tạm thời)
            console.warn("Stats error:", data.message);
        }
    } catch (error) {
        console.log("Polling skipped:", error.message);
    } finally {
        // QUAN TRỌNG: Chỉ gọi lần tiếp theo sau khi lần này đã xong (hoặc
```

```
lỗi)
    // Đợi 3 giây (3000ms) để giảm tải cho server
    if (document.querySelector('[data-page="home"]')) {
        pollingTimer = setTimeout(fetchStats, 3000);
    }
}

function updateDashboardData(data) {
    const setText = (id, val) => {
        const el = document.getElementById(id);
        if(el) el.textContent = val;
    };

    setText('dash-hostname', data.hostname);
    setText('dash-os', data.os_info);
    setText('dash-ip', data.internal_ip);
    setText('dash-latency', data.latency);
    setText('dash-ram', data.ram_free);
    setText('dash-battery', data.battery);

    const cpu = parseFloat(data.cpu_load);
    setText('dash-cpu', cpu);

    const bar = document.getElementById('dash-cpu-bar');
    if(bar) {
        bar.style.width = `${Math.min(cpu, 100)}%`;
        let colorClass = "bg-blue-600";
        if(cpu > 80) colorClass = "bg-red-600";
        else if(cpu > 50) colorClass = "bg-yellow-500";
        bar.className = `${colorClass} h-2.5 rounded-full transition-all duration-500`;
    }
}

// --- KHỞI CHẠY ---
// Kiểm tra ngay trạng thái localStorage để render UI ngay lập tức (không chờ request đầu tiên)
const savedServer = localStorage.getItem('connectedServer');
if (savedServer) {
    setUI(true); // Hiện dashboard ngay (loading state)
    fetchStats(); // Bắt đầu polling
} else {
    setUI(false);
}
}

// Đăng ký sự kiện HTMX và DOMContentLoaded
document.addEventListener('htmx:afterSwap', initHomePage);
document.addEventListener('DOMContentLoaded', initHomePage);
</script>
```

## 2. Sửa `socket_client_persistent.py` (Chống treo Socket)

Cần đảm bảo hàm `get_system_stats` có timeout cực ngắn. Nếu C# server bị treo, Python phải nhả ngay lập tức để Django còn xử lý việc khác.

Tìm hàm `get_system_stats` và sửa lại phần `try/except` (thêm timeout cho socket):

```
def get_system_stats(self):
    if not self.connected:
        return {"status": "error", "message": "Not connected"}

    with self._lock:
        try:
            # --- THÊM TIMEOUT TẠM THỜI ---
            # Chỉ chờ C# trả lời trong 2 giây. Nếu lâu hơn -> Bỏ qua.
            self.socket.settimeout(2.0)

            start_time = time.time()

            self._send_str("SYSTEM_INFO")
            response = self._recv_line()

            end_time = time.time()

            # --- TRẢ LẠI TIMEOUT MẶC ĐỊNH ---
            self.socket.settimeout(self.timeout)

            # ... (Phần xử lý latency và data giữ nguyên) ...
            latency = round((end_time - start_time) * 1000, 0)

            # ... (Code parse data giữ nguyên) ...

            if response.startswith("ERROR"):
                return {"status": "error", "message": response}

            parts = response.split('|')
            def get_part(idx, default="?"): return parts[idx] if len(parts) >
idx else default

            data = {
                "cpu_load": get_part(0, "0"),
                "ram_free": get_part(1, "0"),
                "battery": get_part(2, "Unknown"),
                "hostname": get_part(3, "Unknown"),
                "os_info": get_part(4, "Unknown"),
                "internal_ip": get_part(5, "Unknown"),
                "latency": latency
            }

            self._send_str("QUIT")
            return {"status": "success", "data": data}
```

```
except socket.timeout:  
    # Nếu timeout -> Coi như server bận, không ngắt kết nối  
    # Trả về status 'busy' để frontend biết mà không báo lỗi  
    return {"status": "error", "message": "Timeout (Server Busy)"}  
  
except Exception as e:  
    self.connected = False  
    return {"status": "error", "message": str(e)}
```

Tóm tắt thay đổi:

1. **HTML/JS:** Chuyển sang dùng `setTimeout` (gọi lại sau khi xong) thay vì `setInterval` (gọi bất chấp). Điều này ngăn chặn việc dồn ứ hàng trăm request nếu mạng chậm -> **Hết lag.**
2. **HTML/JS:** Logic UI kiểm tra `localStorage` ngay đầu hàm `init`, giúp hiển thị Dashboard ngay cả khi chưa có data mới -> **Hết lỗi refresh mất trạng thái.**
3. **Python:** Thêm `socket.settimeout(2.0)` cho hàm stats. Nếu Server C# đơ quá 2 giây, Python sẽ bỏ qua và trả về lỗi nhẹ, không treo cả web server Django -> **Web mượt hơn.**

Prompt:

navigation - HTML

Ok! Trang home đã chạy và khá bình thường. Chỉ có điều có hơi ít thông tin không nhỉ? Tất cả mấy cái thông tin performance trong task manager không bung hết ra được hả? Với cả thông tin phần cứng, model của máy thì sao?

Nếu rắc rối quá thì thôi đừng bưng, nhưng nếu dễ, chỉ cần gọi thư viện thì cứ bưng vào.

Với lại có thêm lỗi này không liên quan đến homepage, đôi khi bấm vào nút connect thì nút đó nhảy từ connect -> disconnect -> disconnecting, giống như kiểu đã bị bấm liên tục cho đến khi đã kích hoạt disconnect mặc dù vẫn chưa kịp connect vào server.

Bình thường thì tôi nhớ bấm connect xong sẽ có Connecting... để tránh trường hợp này xảy ra...

Response:

Thinking:

Thinking:

### Exploring System Insights

I've been meticulously analyzing the user's request. My focus is on understanding their desire for enhanced system insights on the dashboard. I'm prioritizing the integration of Task Manager-esque performance metrics, detailed hardware information, and real-time resource utilization displays. Currently, I'm working out how to most elegantly present this data without overwhelming the interface.

### Addressing UX Hiccup

I'm now prioritizing the UI issue. The erratic "Connect" button behavior is my main focus. I believe the solution lies in improving state management during the connection process. I suspect a race condition, so I am thinking of how to block input or lock the UI during the "Connecting..." state. I'll need to review the `navigation.html` file closely to understand the button's event handling. I am also planning on updating the system information display soon.

## Prioritizing Resource Monitoring

I'm now prioritizing the system information display. I am aiming to incorporate the following: Total RAM with `Microsoft.VisualBasic.Devices.ComputerInfo`, GPU details using WMI's `Win32_VideoController`, Disk Usage with a `PerformanceCounter`, and Up Time via `Environment.TickCount`. I have started to update `server.cs` to include the fetching and display of static hardware data, as well as the implementation of counters for Disk usage, focusing on simplicity and avoiding network complexity for now.

## Investigating Button Behavior

I'm zeroing in on the button issue. The "Connect -> Disconnect -> Disconnecting" sequence is concerning. My hypothesis is now that rapid swapping of the buttons after connection is the root cause. I am currently focusing on implementing strict disabling of the `connectBtn` immediately upon click. I am also planning on adding a delay before enabling the `disconnectBtn` to provide the user with a better interface. I will also be updating the system information display soon.

## Analyzing Connection Flow

I'm now zeroing in on connection flow. After reviewing the code, it seems the `finally` block in the connection process is causing `connectBtn` to stay disabled. I need to ensure it remains disabled throughout. The rapid switching suggests the `disconnectBtn` immediately appears. I'm investigating potential conflicts between `restoreConnectionUI` and the click handler, possibly due to race conditions. I'm focusing on rigorously disabling the `connectBtn` and carefully controlling the state transitions to prevent any unexpected behavior. I also plan to verify when and why `restoreConnectionUI` is triggered.

## Defining Connection State

I've decided to tackle the connection button issue directly. My key strategy is to use the `isConnecting` flag to prevent concurrent executions. Also, I'll make sure the `disconnectBtn` isn't interactive immediately, which should help mitigate the UI's quick swapping after connection. I'll need to go through the steps of fetching static and dynamic info, updating the pipe string format and adapting the client, and updating UI elements to display the new information. I will be sure to address issues in the connection state management.

## Adding Hardware Details

I'm now focusing on displaying hardware details. I'll need to explicitly guide the user to add the `System.Management` reference in their C# project. I'm also adding WMI calls to get processor information, and implementing a disk usage counter. I need to make sure the server and client communicate correctly to send and display this data.

Được, mình sẽ nâng cấp Dashboard để hiển thị thông tin chi tiết như Task Manager (Disk usage, GPU name, CPU model...) và sửa lỗi nút Connect bị "nhảy" lung tung.

Để lấy thông tin phần cứng (Tên CPU, GPU, Tổng RAM), chúng ta cần dùng thư viện **System.Management** trong C#. Nó có sẵn trong Windows nhưng mặc định Visual Studio không tham chiếu đến nó.

Dưới đây là các bước thực hiện:

## BƯỚC 1: Cập nhật **server.cs** (Thêm thông tin phần cứng)

**⚠ QUAN TRỌNG:** Trong Visual Studio, bạn cần thêm Reference:

1. Chuột phải vào **References** trong Solution Explorer -> **Add Reference...**
2. Chọn tab **Assemblies** -> **Framework**.
3. Tìm và tick chọn **System.Management**.
4. Bấm OK.

Sau đó thay thế code trong **server.cs**:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;
using Microsoft.Win32;
using System.IO;
using System.Diagnostics;
using System.Threading;
using System.Management; // <--- NHỚ THÊM REFERENCE NÀY

namespace ServerApp
{
    public partial class server : Form
    {
        // ... (Các biến thread cũ giữ nguyên) ...

        System.Timers.Timer statsTimer;
        string cachedSystemInfo = "";

        // Counters
        PerformanceCounter cpuCounter;
        PerformanceCounter ramCounter;
        PerformanceCounter diskCounter; // Mới: Disk Usage

        // Static Info (Lấy 1 lần lúc bật app thôi)
        string staticHardwareInfo = "";
    }
}
```

```
public server()
{
    InitializeComponent();
    CheckForIllegalCrossThreadCalls = false;
    this.FormClosing += new FormClosingEventHandler(server_FormClosing);

    // 1. Lấy thông tin tĩnh (Model CPU, GPU, Total RAM)
    Thread t = new Thread(GetStaticHardwareInfo);
    t.IsBackground = true;
    t.Start();

    // 2. Khởi tạo Counters
    try {
        cpuCounter = new PerformanceCounter("Processor", "% Processor
Time", "_Total");
        ramCounter = new PerformanceCounter("Memory", "Available MBytes");
        diskCounter = new PerformanceCounter("PhysicalDisk", "% Disk
Time", "_Total"); // Disk Usage

        // Warm up
        cpuCounter.NextValue();
        diskCounter.NextValue();

        statsTimer = new System.Timers.Timer(1000);
        statsTimer.Elapsed += UpdateSystemStats;
        statsTimer.Start();
    } catch { }
}

// Hàm lấy thông tin phần cứng (Chạy 1 lần)
private void GetStaticHardwareInfo()
{
    try {
        string cpuName = "Unknown CPU";
        string gpuName = "Unknown GPU";
        string totalRam = "0";

        // Lấy CPU Name
        using (var searcher = new ManagementObjectSearcher("select Name
from Win32_Processor")) {
            foreach (var item in searcher.Get()) { cpuName =
item["Name"].ToString(); break; }
        }

        // Lấy GPU Name
        using (var searcher = new ManagementObjectSearcher("select Name
from Win32_VideoController")) {
            foreach (var item in searcher.Get()) { gpuName =
item["Name"].ToString(); break; }
        }

        // Lấy Total RAM (GB)
        using (var searcher = new ManagementObjectSearcher("select
Capacity from Win32_PhysicalMemory")) {
```

```
        long totalBytes = 0;
        foreach (var item in searcher.Get()) { totalBytes +=
Convert.ToInt64(item["Capacity"]); }
        totalRam = (totalBytes / (1024 * 1024 * 1024)).ToString() + "GB";
    }

    staticHardwareInfo = $"{cpuName}|{gpuName}|{totalRam}";
} catch { staticHardwareInfo = "Generic CPU|Generic GPU|Unknown RAM";
}
}

private void UpdateSystemStats(Object source,
System.Timers.ElapsedEventArgs e)
{
    try
    {
        float cpu = cpuCounter.NextValue();
        float ramFree = ramCounter.NextValue();
        float disk = diskCounter.NextValue(); // Lấy % Disk
        if (disk > 100) disk = 100; // Đôi khi nó vọt lõi

        // Pin
        String battery = "Unknown";
        try {
            PowerStatus pwr = SystemInformation.PowerStatus;
            battery = (pwr.BatteryLifePercent * 100).ToString() + "%";
            if (pwr.PowerLineStatus == PowerLineStatus.Online) battery += "(Plugged)";
        } catch {}

        String hostname = Dns.GetHostName();
        String os = Environment.OSVersion.ToString();

        // Uptime (Giờ:Phút)
        TimeSpan uptimeSpan =
TimeSpan.FromMilliseconds(Environment.TickCount);
        string uptime = string.Format("{0}d {1}h {2}m", uptimeSpan.Days,
uptimeSpan.Hours, uptimeSpan.Minutes);

        // Lấy IP
        String ipAddr = "Unknown";
        try {
            var host = Dns.GetHostEntry(Dns.GetHostName());
            foreach (var ip in host.AddressList) {
                if (ip.AddressFamily == AddressFamily.InterNetwork) {
ipAddr = ip.ToString(); break; }
            }
        } catch {}

        // FORMAT MỚI:
        //
CPU|RAM_FREE|BATTERY|HOSTNAME|OS|IP|DISK|UPTIME|CPU_MODEL|GPU_MODEL|TOTAL_RAM
cachedSystemInfo = $"{cpu:0.0}|{ramFree}|{battery}|{hostname}|
```

```

{os}|{ipAddr}|{disk:0.0}|{uptime}|{staticHardwareInfo}";
}
catch { }
}

public void send_system_info()
{
    try
    {
        Program.nw.WriteLine(cachedSystemInfo);
        Program.nw.Flush();
    }
    catch (Exception ex)
    {
        Program.nw.WriteLine("ERROR|" + ex.Message);
        Program.nw.Flush();
    }
}

// ... (Giữ nguyên các phần khác) ...
}
}

```

## BƯỚC 2: Cập nhật `socket_client_persistent.py` (Python)

Cập nhật hàm parse dữ liệu để đọc thêm các trường mới.

```

def get_system_stats(self):
    # ... (giữ nguyên đoạn check connect và lock) ...
    with self._lock:
        try:
            self.socket.settimeout(2.0)
            self._send_str("SYSTEM_INFO")
            response = self._recv_line()
            self.socket.settimeout(self.timeout)

            # ... (giữ nguyên logic ping) ...

            parts = response.split('|')
            def get_part(idx, default="?"): return parts[idx] if len(parts) >
idx else default

            # Mapping dữ liệu mới
            data = {
                "cpu_load": get_part(0, "0"),
                "ram_free": get_part(1, "0"),
                "battery": get_part(2, "Unknown"),
                "hostname": get_part(3, "Unknown"),
                "os_info": get_part(4, "Unknown"),
                "internal_ip": get_part(5, "Unknown"),
                "disk_usage": get_part(6, "0"),      # Mới

```

```

        "uptime": get_part(7, "0h 0m"),           # Mới
        "cpu_model": get_part(8, "Generic CPU"), # Mới
        "gpu_model": get_part(9, "Generic GPU"), # Mới
        "total_ram": get_part(10, "? GB")       # Mới
    }

    # Tính toán % RAM đã dùng (Cho UI đẹp hơn)
    try:
        free_mb = float(data["ram_free"])
        total_gb_str = data["total_ram"].split(' ')[0]
        total_mb = float(total_gb_str) * 1024
        if total_mb > 0:
            used_percent = ((total_mb - free_mb) / total_mb) * 100
            data["ram_usage_percent"] = round(used_percent, 1)
        else:
            data["ram_usage_percent"] = 0
    except:
        data["ram_usage_percent"] = 0

    self._send_str("QUIT")
    return {"status": "success", "data": data}

except Exception as e:
    # ... (giữ nguyên xử lý lỗi) ...

```

### BƯỚC 3: Cập nhật `home_partial.html` (Giao diện mới)

Tôi sẽ thiết kế lại giao diện để hiển thị nhiều thông tin hơn (Uptime, Disk, GPU...).

```

<main data-page="home">
    <div class="px-4 pt-6">

        <div id="dash-disconnected" class="hidden">
            <div class="p-8 text-center bg-white border border-gray-200 rounded-lg shadow-sm dark:bg-gray-800 dark:border-gray-700">
                <div class="inline-flex items-center justify-center w-16 h-16 mb-4 bg-gray-100 rounded-full dark:bg-gray-700">
                    <svg class="w-8 h-8 text-gray-400" fill="none" stroke="currentColor" viewBox="0 0 24 24"><path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M12 6v6m0 0h6m-6 0H6" /></path></svg>
                </div>
                <h3 class="text-lg font-medium text-gray-900 dark:text-white">No Active Connection</h3>
                <p class="mt-2 text-gray-500 dark:text-gray-400">Select a server to view detailed metrics.</p>
            </div>
        </div>

        <div id="dash-connected" class="hidden space-y-4">

            <div class="p-4 bg-white border border-gray-200 rounded-lg shadow-sm"

```

```
dark:border-gray-700 dark:bg-gray-800">
    <div class="flex flex-col md:flex-row md:items-center justify-between gap-4">
        <div>
            <h2 class="flex items-center gap-3 text-2xl font-bold text-gray-900 dark:text-white">
                <span class="relative flex w-3 h-3">
                    <span class="absolute inline-flex w-full h-full bg-green-400 rounded-full opacity-75 animate-ping"></span>
                    <span class="relative inline-flex w-3 h-3 bg-green-500 rounded-full"></span>
                </span>
                <span id="dash-hostname">Loading...</span>
            </h2>
            <div class="mt-1 text-sm text-gray-500 dark:text-gray-400">
                <span id="dash-os">Checking OS...</span> • <span id="dash-upptime">Uptime: --</span>
            </div>
        </div>
        <div class="text-right">
            <div class="text-sm text-gray-500 dark:text-gray-400">Latency</div>
            <div class="text-2xl font-bold text-green-500"><span id="dash-latency">--</span> <small>ms</small></div>
        </div>
    </div>

    <hr class="my-4 border-gray-200 dark:border-gray-700">

    <div class="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-4 text-sm">
        <div>
            <span class="block text-gray-500 dark:text-gray-400">Processor (CPU)</span>
            <span id="dash-cpu-model" class="font-medium text-gray-900 dark:text-white">--</span>
        </div>
        <div>
            <span class="block text-gray-500 dark:text-gray-400">Graphics (GPU)</span>
            <span id="dash-gpu-model" class="font-medium text-gray-900 dark:text-white">--</span>
        </div>
        <div>
            <span class="block text-gray-500 dark:text-gray-400">Memory (RAM)</span>
            <span id="dash-total-ram" class="font-medium text-gray-900 dark:text-white">--</span>
        </div>
        <div>
            <span class="block text-gray-500 dark:text-gray-400">Network IP</span>
            <span id="dash-ip" class="font-medium text-gray-900 dark:text-white">--</span>
        </div>
    </div>

```

```
dark:text-white">--</span>
    </div>
</div>
</div>

<div class="grid grid-cols-1 gap-4 md:grid-cols-3">

    <div class="p-4 bg-white border border-gray-200 rounded-lg shadow-sm dark:bg-gray-800 dark:border-gray-700">
        <div class="flex justify-between mb-2"><span class="text-sm font-medium text-gray-500 dark:text-gray-400">CPU Usage</span></div>
        <div class="flex items-end gap-2">
            <span id="dash-cpu" class="text-4xl font-bold text-gray-900 dark:text-white">0</span><span class="mb-1 text-gray-500">%</span>
        </div>
        <div class="w-full h-2 mt-3 bg-gray-200 rounded-full dark:bg-gray-700">
            <div id="dash-cpu-bar" class="h-2 bg-blue-600 rounded-full transition-all duration-500" style="width: 0%"></div>
        </div>
    </div>

    <div class="p-4 bg-white border border-gray-200 rounded-lg shadow-sm dark:bg-gray-800 dark:border-gray-700">
        <div class="flex justify-between mb-2">
            <span class="text-sm font-medium text-gray-500 dark:text-gray-400">Memory</span>
            <span id="dash-ram-txt" class="text-xs text-gray-400">-- / --</span>
        </div>
        <div class="flex items-end gap-2">
            <span id="dash-ram-percent" class="text-4xl font-bold text-gray-900 dark:text-white">0</span><span class="mb-1 text-gray-500">%</span>
        </div>
        <div class="w-full h-2 mt-3 bg-gray-200 rounded-full dark:bg-gray-700">
            <div id="dash-ram-bar" class="h-2 bg-purple-600 rounded-full transition-all duration-500" style="width: 0%"></div>
        </div>
    </div>

    <div class="p-4 bg-white border border-gray-200 rounded-lg shadow-sm dark:bg-gray-800 dark:border-gray-700">
        <div class="flex justify-between mb-2"><span class="text-sm font-medium text-gray-500 dark:text-gray-400">Disk Active Time</span></div>
        <div class="flex items-end gap-2">
            <span id="dash-disk" class="text-4xl font-bold text-gray-900 dark:text-white">0</span><span class="mb-1 text-gray-500">%</span>
        </div>
        <div class="w-full h-2 mt-3 bg-gray-200 rounded-full dark:bg-gray-700">
            <div id="dash-disk-bar" class="h-2 bg-green-500 rounded-full transition-all duration-500" style="width: 0%"></div>
        </div>
    </div>

```

```
        </div>
    </div>

    <div class="p-4 bg-white border border-gray-200 rounded-lg shadow-sm dark:bg-gray-800 dark:border-gray-700">
        <div class="flex items-center justify-between">
            <div>
                <span class="text-sm font-medium text-gray-500 dark:text-gray-400">Battery Status</span>
                <div class="text-lg font-bold text-gray-900 dark:text-white" id="dash-battery">--</div>
            </div>
            <div class="p-3 bg-blue-100 rounded-full dark:bg-blue-900">
                <svg class="w-6 h-6 text-blue-600 dark:text-blue-300" fill="none" stroke="currentColor" viewBox="0 0 24 24"><path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M13 10V3L4 14h7v7l9-11h-7z"></path>
            </svg>
        </div>
    </div>
</main>

<script>
    function initHomePage() {
        if (!document.querySelector('[data-page="home"]')) return;

        const viewDisconnected = document.getElementById('dash-disconnected');
        const viewConnected = document.getElementById('dash-connected');
        let pollingTimer = null;

        function setUI(isConnected) {
            if (isConnected) {
                viewDisconnected.classList.add('hidden');
                viewConnected.classList.remove('hidden');
            } else {
                viewConnected.classList.add('hidden');
                viewDisconnected.classList.remove('hidden');
            }
        }

        async function fetchStats() {
            if (!document.querySelector('[data-page="home"]')) return;

            try {
                if (!localStorage.getItem('connectedServer')) {
                    setUI(false);
                    return;
                }

                const response = await fetch('/remote/api/stats/');
                if (!response.ok) throw new Error("Busy");
            }
        }
    }
</script>
```

```
        const data = await response.json();

        if (data.status === 'success') {
            setUI(true);
            updateData(data.data);
        } else if (data.status === 'disconnected') {
            setUI(false);
        }

    } catch (error) {
    // Ignore
} finally {
    if (document.querySelector('[data-page="home"]')) {
        pollingTimer = setTimeout(fetchStats, 2000);
    }
}

function updateData(d) {
    const setText = (id, val) => { const el = document.getElementById(id);
if(el) el.textContent = val; };

    setText('dash-hostname', d.hostname);
    setText('dash-os', d.os_info);
    setText('dash-ip', d.internal_ip);
    setText('dash-latency', d.latency);
    setText('dash-battery', d.battery);

    setText('dash-uptime', "Uptime: " + d.uptime);
    setText('dash-cpu-model', d.cpu_model);
    setText('dash-gpu-model', d.gpu_model);
    setText('dash-total-ram', d.total_ram);

    // CPU
    const cpu = parseFloat(d.cpu_load);
    setText('dash-cpu', cpu);
    document.getElementById('dash-cpu-bar').style.width = `${Math.min(cpu,
100)}%`;

    // RAM (dùng percent tính từ Python)
    const ramPercent = d.ram_usage_percent || 0;
    setText('dash-ram-percent', ramPercent);
    document.getElementById('dash-ram-bar').style.width =
`${Math.min(ramPercent, 100)}%`;
    // Hiện Free/Total
    const totalGb = parseFloat(d.total_ram) || 0;
    const freeMb = parseFloat(d.ram_free) || 0;
    const usedGb = totalGb - (freeMb/1024);
    setText('dash-ram-txt', `${usedGb.toFixed(1)}GB used /
${d.total_ram}`);

    // Disk
    const disk = parseFloat(d.disk_usage);
```

```

        setText('dash-disk', disk);
        document.getElementById('dash-disk-bar').style.width =
` ${Math.min(disk, 100)}%`;
    }

    const savedServer = localStorage.getItem('connectedServer');
    if (savedServer) {
        setUI(true);
        fetchStats();
    } else {
        setUI(false);
    }
}

document.addEventListener('htmx:afterSwap', initHomePage);
document.addEventListener('DOMContentLoaded', initHomePage);
</script>

```

## BƯỚC 4: Sửa lỗi nút Connect bị "nhảy" ([navigation.html](#))

Vấn đề nút bị nhảy "Connect -> Disconnect -> Disconnecting" là do sau khi kết nối thành công, nút **Connect** bị ẩn và nút **Disconnect** hiện ra **ngay tại vị trí con trỏ chuột**. Nếu người dùng click đúp hoặc click nhanh, click thứ 2 sẽ trúng nút Disconnect vừa hiện ra.

Cách sửa: Thêm logic ngăn chặn tương tác nút Disconnect trong vài giây đầu sau khi connect.

Trong file [navigation.html](#), tìm đến phần Script **DOMContentLoaded** và sửa 2 chỗ:

### 1. Sửa sự kiện **connectBtn**:

```

// Trong sự kiện connectBtn.click
// ...
if (data.success) {
    showNotification('success', `Connected to ${selectedIp}`);

    // ... (lưu trạng thái) ...

    // Fix lỗi UI Jump: Disable nút Disconnect tạm thời
    disconnectBtn.disabled = true;

    // Đổi nút
    connectBtn.classList.add('hidden');
    disconnectBtn.classList.remove('hidden');

    // Bật lại nút Disconnect sau 1.5 giây (để tránh double-click nhầm)
    setTimeout(() => {
        disconnectBtn.disabled = false;
    }, 1500);

} else {

```

```
// ...
}
```

**2. Sửa sự kiện disconnectBtn:** Thêm kiểm tra an toàn để đảm bảo không gửi lệnh khi nút đang bị disable (dù HTML disable nhưng thêm check JS cho chắc).

```
if (disconnectBtn) {
    disconnectBtn.addEventListener('click', async function() {

        // --- THÊM CHECK NÀY ---
        if (disconnectBtn.disabled) return;
        // -----

        const activeStream = localStorage.getItem('ACTIVE_STREAM_MODE');
        // ... (phần còn lại giữ nguyên)
```

Bạn hãy áp dụng các thay đổi trên. Bây giờ Dashboard sẽ hiển thị rất chuyên nghiệp với thông tin phần cứng tĩnh (Hardware) và thông số động (Performance) mà không gây lag. Nút Connect cũng sẽ "lì" hơn, không bị nhảy lung tung nữa.

---

Powered by [Gemini Exporter](#)