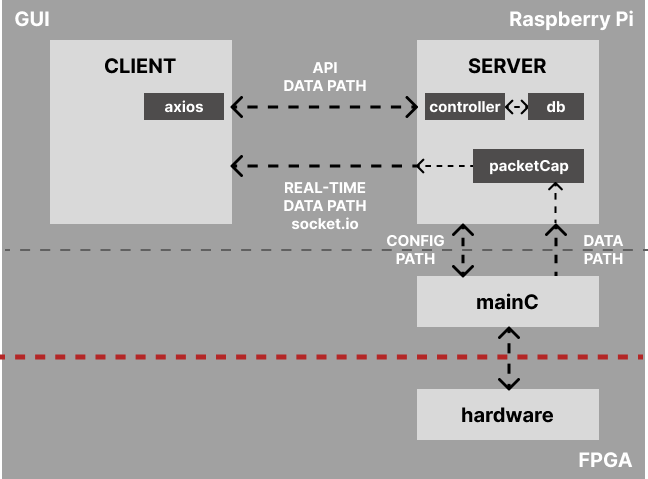
**SPECIFICATION GUI ANTIDDOS 1GB**

1. **Kiến trúc tổng quan GUI AntiDDoS 1Gb.**



Ngôn ngữ sử dụng: **Javascript**

GUI của AntiDDoS - 1Gb bao gồm 2 Code-base. 1 Code-base của server (Express.js) và 1 của client (React.js).

* **Server:** RESTful API Server.

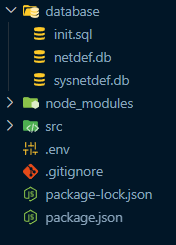
Thao tác với CSDL, cấu hình phần cứng thông qua MainC, nhận packet từ hardware đẩy lên thông qua mainC và format lại để hiển thị lên GUI.

* Client: Single Page Application (**SPA**), Client Side Rendering (**CSR**)

Hiển thị trực quan các lưu lượng hiện tại của mạng qua thiết bị. Dữ liệu real-time được truyền nhận giữa server - client thông qua Socket.IO.

1. **Kiến trúc thư mục dự án.**
   1. **Server**

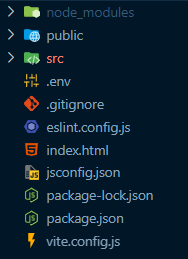
* Framework: Express.js (Node.js) + Socket.IO

****

* Database/: Thư mục chứa SQLite DB của server (**sysnetdef.db**) và script khởi tạo DB (**init.sql)**
* node\_modules/: Thư mục chứa các thư viện của project được node package manager (**npm**) quản lý, khi mới lấy source code về, node\_modules sẽ không có mà cần phải cài các thư viện của project thông qua câu lệnh

npm install

* src/: Chứa source code của project. Chi tiết mô tả trong mục 3.
* .env: File môi trường của project. Sẽ thay đổi tùy thuộc vào nơi chạy project. Devlopment (máy dev) và Product (Raspberry Pi)
* .gitignore: Các file được lưu trong file này sẽ không được đẩy lên git, việc này tránh trường hợp đẩy các file nặng hoặc kh cần thiết lên git.
* package.json: Chứa các khai báo thư viện cho project, lúc chạy npm install , lệnh sẽ dựa vào file trên để tiến hành cài các thư viện cần thiết cho project. Ngoài ra, đây là nơi mà chúng ta khai báo các script để chạy project khi debug
  1. **Client**
* Framework: Vite + ReactJs cùng với Ant Design UI Material

****

* node\_modules/: Chứa tất cả các thư viện (dependencies) mà dự án cài thông qua npm hoặc. Thư mục này thường được tạo tự động sau khi chạy npm install.
* public/: Chứa các file tĩnh như hình ảnh, favicon, fonts,... Các file ở đây sẽ được phục vụ trực tiếp mà không cần thông qua bộ biên dịch.
* src/: Nơi chứa toàn bộ mã nguồn chính (code) của dự án: các component, page, style, hooks, store,... Tất cả code chính sẽ nằm ở đây.
* .env: File chứa các biến môi trường (environment variables), ví dụ như URL API, key, secret,... Những biến này có thể được truy cập trong code qua import.meta.env.
* .gitignore: Quy định các file/thư mục nào sẽ bị Git bỏ qua, không đưa lên repository, ví dụ: node\_modules, file .env, thư mục dist,...
* eslint.config.js: File cấu hình ESLint để kiểm tra lỗi cú pháp, style code,... đảm bảo code nhất quán và dễ đọc.
* index.html: File HTML chính của ứng dụng. Khi build, Vite sẽ chèn bundle script vào file này. Đây là điểm khởi đầu để trình duyệt tải trang.
* jsconfig.json: Dùng để cấu hình đường dẫn import cho project JavaScript (ví dụ hỗ trợ alias), giúp IntelliSense (tự động hoàn thành code) hoạt động tốt hơn.
* package-lock.json: Tự động tạo ra bởi npm. Ghi lại chi tiết từng phiên bản package đã cài, đảm bảo khi deploy hoặc cài lại, các gói thư viện không bị sai lệch.
* package.json: File cực kỳ quan trọng. Chứa metadata của project (tên, phiên bản, mô tả), danh sách dependencies, script chạy build/dev/test,...
* vite.config.js: File cấu hình chính cho Vite (bộ bundler). Dùng để tuỳ chỉnh các hành vi như alias, plugin, server proxy, build settings,...

1. **Các tính năng chính**

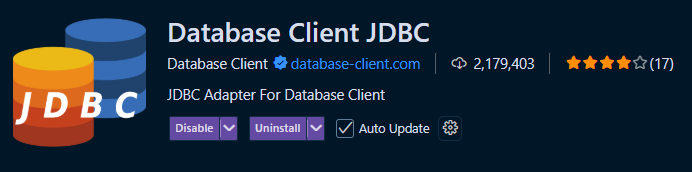
Các chức năng của GUI được liệt kê theo giao diện bao gồm:

* Giám sát lưu lượng mạng mà thiết bị đang xử lý ở hiện tại và quá khứ: **Dashboard, Analyze, Monitor.** Bao gồm các thông tin như tổng thông lượng vào/ra thiết bị, tổng thông lượng bypass/drop, thông lượng theo protocol, thông lượng theo giao thức tấn công,...
* **(INTERFACE)** Cấu hình chế độ hoạt động cho thiết bị: **Defense.** 
  + (**General)** Cho phép cấu hình chế độ bảo vệ của các port (interface) trên thiết bị, bao gồm:
    - Chỉ định Port nào là port server
    - Chế độ bảo vệ theo port & theo địa chỉ IP
    - Cấu hình Profile bảo vệ
  + **(Defense Profiles)** Quản lý các profile cấu hình: trong profile sẽ chứa các ngưỡng để cấu hình xuống dưới phần cứng để phần cứng hoạt động
  + **(Port mirroring)** Cấu hình port mirroring: Cấu hình port để sao chép gói tin từ port được chỉnh định mirroring sang port monitoring
  + Quản lý Access Control List (ACL):
    - **Network protection:** Địa chỉ các IP được bảo vệ tương ứng với interface
    - **Attacker List:** Địa chỉ IP tấn công (IP đích), các địa chỉ ở trong danh sách này sẽ ngay lập tức bị đánh chặn
    - **VPN Whitelist:** Địa chỉ IP trong danh sách VPN sẽ được bypass qua thiết bị
    - **HTTP Attacker list:** Địa chỉ IP trong danh sách trên sẽ bị đánh chặn đối với giao thức HTTP
* **(Management)** Quản lý thiết bị (Đối với tài khoản Admin)
  + (**Users)** Quản lí danh sách người dùng
    - Thêm/Xóa/Sửa
    - Quản lý các sự kiện gởi mail của người dùng khi có tấn công ddos, tấn công kết thúc, đầy ổ đĩa
  + **(Activities & Log)** Quản lí các sự kiện, log:
    - Log:
      * Tải/Xóa log
      * Tự động xóa log, set ngưỡng tự động xóa,...
    - Sự kiện:
      * Sự kiện cấu hình: liên quan đến các hành động cấu hình xuống phần cứng.
      * Sự kiện hệ thống: liên quan đến các hoạt động diễn ra như thêm/xóa người dùng,...
  + **(Device)** Theo dõi tình trạng sử dụng tài nguyên của thiết bị
* Đối với tài khoản người dùng thông thường, có thể tự cập nhật thông tin cá nhân tại mục **PROFILE**
  1. **Kiến trúc CSDL**

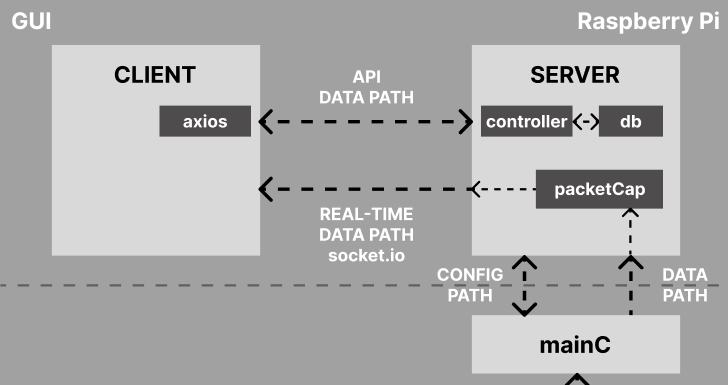
****

* **Các bảng trong CSDL:**
  + **Users:** Lưu trữ thông tin của user, và các option cho việc gởi mail tương ứng với từng User đó
  + **Device Interfaces:** Lưu trữ thông tin của các port trên thiết bị
  + **Network addresses:** Lưu trữ & quản lý Access Control List
  + **Defense profiles:** Lưu trữ các profile cấu hình thông số của phần cứng
  + **Device Logs:** Lưu trữ các **Activity** của thiết bị
  + **Device Settings:** Lưu trữ các cài đặt của thiết bị bao gồm (bật tắt chế độ tự động xóa, cài đặt ngưỡng)
  + **Network Anomalies:** Lưu trữ các cuộc tấn công DDoS (hiển thị trang Dashboard)

\* Để có thể tương tác với CSDL (**sysnetdef.db)** ta sử dụng VSCode Extension:

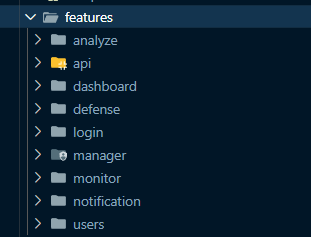


* 1. **Triển khai**

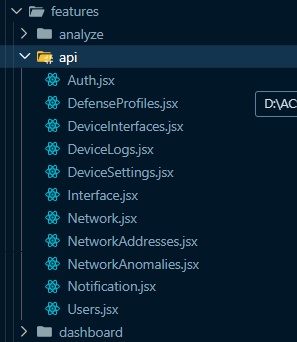
****

Vì Server là 1 **RESTful API Server** cho nên server sẽ không trả về cho ta bất kì giao diện (HTML) nào cả mà chỉ trả về data cho client thông qua JSON format.

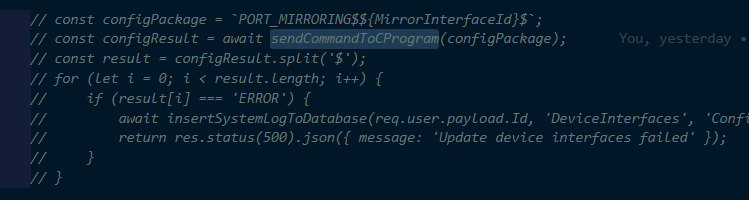
* Client nhận thông tin từ server thông qua 2 đường:
  + API:
    - **Đối với CLIENT**



* Client được xây dựng từ các features. Bên trong các features sẽ là các component. 1 Trang sẽ được hình thành từ các component bên trong các features tương ứng. Client triển khai theo **Client Side Rendering (CSR)**. Cho nên khi các component được load (mount), nó sẽ tự động gọi API đến server để lấy thông tin thông qua **REACT hooks.**

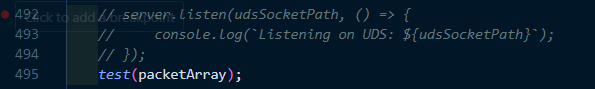
****

* Ở trong các hooks, ta sẽ gọi các API được định nghĩa ở trong thư mục trên để lấy dữ liệu từ SERVER đối với đối tượng tương ứng.
* **Đối với SERVER.**
* Server được triển khai theo mô hình MVC (Model - Views - Control), tuy nhiên không có Views mà thay vào đó là **SPA.**
* Ứng với mỗi models, ta sẽ có một controller tương ứng để tương tác với chúng.
* Các api routes dùng cho client tương tác với controller nằm trong routes
  + REAL-TIME: Server nhận thông tin từ mainC **(packetCap.service)** đẩy lên thông qua **uds socket**, sau khi xử lý thông tin thì chuyển lên cho client thông qua socket.io
  1. **Các services đặt biệt (ở server)**
* **socket.service:** Được sử dụng để gởi gói tin cấu hình xuống mainC (phần cứng). Service trên được sử dụng tại rải rác các controller có công dụng cấu hình **(mục interface)**.



Đoạn code trên là một ví dụ về socket.service được dùng tại các controller, ở trong môi trường development, các đoạn code trên sẽ được comment lại do không có phần cứng để tương tác, vậy nên trong quá trình test đọc/ghi vào CSDL sẽ bị comment, khi cập nhật code lên môi trường product cần mở ra

* **packetCap.service:** nhận gói tin, phân tích và đẩy data về client qua socket.io để vẽ biểu đồ

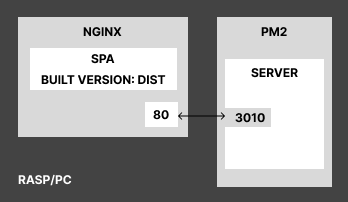


Tương tự như socket.service, khi nằm trong môi trường dev, ta sẽ comment dòng lệnh 492 - 494 để không bị lỗi. Khi cần bơm gói tin ảo vào ta sẽ mở comment cho dòng 495

* **logAnalyze.service:** Chứa các source code liên quan đến đọc logfile để vẽ biểu đồ tại trang analyze
* **email.service:** Chứa các logic code liên quan đến gởi email đến email của người dùng.

1. **FULL Deploy**

Khi deploy và chuẩn bị cho môi trường 100% production. Ta cần cài các thư viện, cũng như tools để hỗ trợ cho việc vận hành server trên thiết bị một các trơn tru. Để setup server, ta cần 2 thứ. **PM2** và **NGINX**

****

**NGINX:**

* NGINX sẽ phục vụ SPA (Single Page Application) từ thư mục build (**dist/** hoặc **build/**). Lúc này NGINX sẽ đóng vai trò như một vai trò thứ 2, có trách nhiệm gởi file chứa giao diện (SPA) cho máy của người dùng truy cập đến
* Đồng thời, NGINX làm proxy: nếu nhận các request API thì nó sẽ chuyển **proxy\_pass** về server Express.

**PM2:**

* PM2 quản lý SERVER, tức là Express app đang chạy trên port 3010.
* PM2 giữ cho app tự restart nếu crash, và giám sát uptime.

**Flow dữ liệu:**

* Người dùng truy cập port 80 (mặc định HTTP) → NGINX xử lý.
* Nếu là request lấy file tĩnh (HTML, JS, CSS) → trả trực tiếp từ thư mục SPA
* Nếu là request API (ví dụ /api/user) → NGINX chuyển tiếp nội bộ sang Express server qua cổng 3010.

### **4.1. Cài đặt PM2 cho SERVER**

* PM2 là một process manager chuyên dụng để chạy ứng dụng Node.js nền production.

**4.1.1. Cài đặt: npm install -g pm2**

### **4.1.2. Chạy server thông qua pm2 với lệnh:**

### **pm2 start src/app.js --watch --name sysnetdef --log-date-format 'DD-MM-YY HH:mm:ss.SSS'**

### **4.1.3. Cấu hình tự động restart khi máy chủ reboot**

* Lưu trạng thái: **pm2 save**
* Tạo script tự động chạy: **pm2 startup**
* Sau khi chạy startup, chương trình sẽ gen ra một câu lệnh script, copy-paste và chạy nó trên chính cửa sổ terminal hiện tại để setup auto run

### **4.2. Build project React**

Tại thư mục code client, ta chạy **npm run build**

**Kết quả:** một thư mục **dist/** chứa toàn bộ source code đã được tối ưu thành 1 file index cùng các file css, js khác

### **4.2.1. Copy build đến nơi server sẽ đọc**

**Location:  
/var/www/html/**

**Copy:**

**sudo cp -r dist/\* /var/www/html/**

## **4.3. Cấu hình Nginx để serve SPA và proxy API**

### **4.3.1. Cài đặt Nginx (nếu chưa có)**

**sudo apt update**

**sudo apt install nginx**

### **4.3.2. Cấu hình Nginx sudo nano /etc/nginx/sites-available/default**

**nginx.conf:   
server {**

**listen 80;**

**server\_name \_; # Hoặc IP nếu bạn chưa có domain**

**root /var/www/html;**

**index index.html index.htm;**

**location / {**

**try\_files $uri /index.html;**

**}**

**location /v1/ {**

**proxy\_pass http://localhost:3010/;**

**proxy\_http\_version 1.1;**

**proxy\_set\_header Upgrade $http\_upgrade;**

**proxy\_set\_header Connection 'upgrade';**

**proxy\_set\_header Host $host;**

**proxy\_cache\_bypass $http\_upgrade;**

**}**

**}**

**Giải thích cấu hình:**

* **root:** Đường dẫn tới thư mục chứa file build của React.
* **location /**: Nếu request trùng file tĩnh (**/main.js, /favicon.ico**), Nginx sẽ trả file đó. Nếu không (ví dụ **/about, /profile**...), Nginx sẽ trả **index.html** (để React Router xử lý route).
* **location /v1/**: Bất cứ request nào bắt đầu bằng **/v1/** sẽ được đẩy (proxy\_pass) về Express server port 3010.

**Chú ý:**

* Express server cần phải biết trước rằng các API URL đều bắt đầu với **/v1/.**
* Nếu server Express không có **/v1/**, bạn cần chỉnh sửa lại hoặc thêm prefix **/v1** vào router trong Express

### **4.3.3. Restart Nginx**

**sudo nginx -t # Kiểm tra lỗi syntax**

**sudo systemctl reload nginx**

### **4.4 QUICK RUN**

### **a. Run mainC and display terminal**

cd DDoS\_V1 && sudo ./gui

### **b. Upload frontend (dist)**

sudo cp -r dist/\* /var/www/html/

### **c. Upload server**

**pm2 start sysnetdef**

### **d. Kill port**

**ps au**

**sudo kill \_ \_ \_ \_**

### **e. socket**

**sudo nano /etc/nginx/sites-available/default**