CHƯƠNG 2. CÔNG CỤ VÀ MÔI TRƯỜNG LẬP TRÌNH MẠNG

Trong chương này, chúng ta sẽ khám phá các công cụ, môi trường cần thiết để phát triển, quản lý các ứng dụng, dịch vụ mạng. Một hiểu biết sâu sắc về các công cụ như Wireshark, Postman và Fiddler, môi trường này sẽ giúp xây dựng các ứng dụng mạng hiệu quả, an toàn. Chúng ta bắt đầu với cách nhìn tổng quan về .NET Framework, sau đó tìm hiểu về các công cụ hỗ trợ quan trọng cho quá trình phát triển và kiểm thử ứng dụng mạng. Tiếp theo, xem xét các thư viện và frameworks phổ biến được sử dụng trong lập trình mạng, cũng như các biện pháp an ninh mạng cần thiết để bảo vệ dữ liệu, thông tin quan trọng của người dùng. Bằng cách hiểu rõ về công cụ, môi trường lập trình mạng, chúng ta có thể phát triển các ứng dụng mạng đáng tin cậy, linh hoạt, đồng thời đảm bảo an toàn và bảo mật cho người dùng.

2.1 Giới thiệu

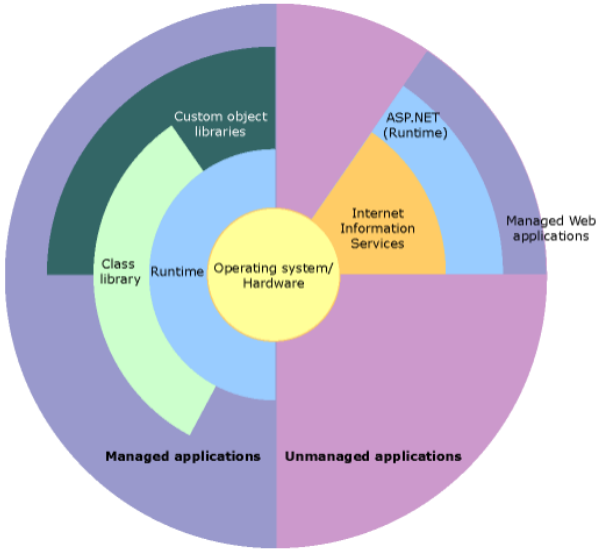
Trong lập trình mạng, sự lựa chọn của công cụ và môi trường là quan trọng. Các công cụ này không chỉ cung cấp một bộ các tính năng lập trình linh hoạt mà còn hỗ trợ lập trình viên trong quá trình xây dựng, triển khai, và duy trì các ứng dụng mạng. Mỗi công cụ lập trình mạng thường đi kèm với một loạt các thư viện và framework giúp tối ưu hóa quy trình phát triển. Điều này giúp lập trình viên tập trung vào logic kinh doanh chính của ứng dụng, không mất nhiều công sức vào những chi tiết thấp cấp của mạng. Trong thời đại ngày nay, với sự phổ cập của Internet, việc sử dụng công cụ và môi trường lập trình mạng là quan trọng không chỉ đối với lập trình viên mạng chuyên nghiệp mà còn đối với các nhà phát triển muốn mở rộng kiến thức của mình vào lĩnh vực này. Sự đa dạng của công cụ và môi trường này mang lại cho cộng đồng lập trình nhiều lựa chọn và cơ hội để đáp ứng nhu cầu đa dạng của dự án mạng từ nhỏ đến lớn. Trong giáo trình này, nhóm tác giả sẽ giới thiệu chi tiết về các công cụ, môi trường lập trình mạng liên quan đến ngôn ngữ C# và nền tảng phát triển ứng dụng mạng là .NET Framework.

2.2 Tổng quan về .NET Framework

Microsoft giới thiệu công nghệ .NET với mục đích cung cấp một nền tảng cho việc đơn giản hóa quá trình phát triển ứng dụng trong môi trường phân bố của Internet. Ứng dụng có thể được phát triển cho các máy trạm và máy chủ bằng nhiều ngôn ngữ lập trình khác nhau. Cùng với .NET, ngôn ngữ lập trình mới C# (C Sharp) được giới thiệu. C# đã trở thành ngôn ngữ lập trình được sử dụng rộng rãi để phát triển các ứng dụng có kết nối mạng cũng như các ứng dụng chạy độc lập.

.NET Framework là một framework để phát triển phần mềm, bao gồm một số lượng lớn các lớp thư viện dành cho việc lập trình, gọi là Framework

Class Library (FCL). Nó hỗ trợ từ việc lập trình giao diện người dùng, truy xuất cơ sở dữ liệu, mã hóa và bảo mật, lập trình web/web service, cũng như lập trình mạng. Có cơ chế cho phép lập trình với ngôn ngữ có thể dùng code được viết bởi các ngôn ngữ khác (language interoperability). Các ứng dụng viết cho .NET Framework sẽ được thực thi trong môi trường CLR (Common Language Runtime).



Hình 2.1 Phân loại của .Net Framework [4]

.NET Framework được thiết kế với các mục đích cung cấp một môi trường lập trình hướng đối tượng nhất quán, cho dù ứng dụng được lưu trữ và thực thi cục bộ, phân bố trên internet, hay lưu trữ và thực thi từ xa. Cung cấp môi trường thực thi mã với: Sự giảm thiểu quá trình phát triển phần mềm và những xung đột phiên bản; thực thi an toàn, bảo gồm các mã được viết bởi các nguồn không rõ ràng hoặc bán tin cậy (semi-trusted third party); việc loại trừ các vấn đề về hiệu năng thực thi mã qua cơ chế thông dịch. Cho phép các nhà phát triển trải nghiệm sự phát triển nhất quán cho nhiều loại ứng dụng đa dạng, từ các ứng dụng Windows-based đến các ứng dụng Web-based. Kết nối với các tiêu chuẩn công nghiệp để đảm bảo rằng mã viết trên .NET Framework có thể tích hợp với các loại mã khác.

CLC (Common Language Runtime) là phần cốt lõi của .NET Framework, dùng để thực thi các chương trình viết trên nền .NET và . NET có những ưu điểm sau:

Cải thiện hiệu năng;

Việc sử dụng lại một cách dễ dàng các components đã được viết trong những ngôn ngữ lập trình khác;

Cung cấp các kiểu dữ liệu có thể mở rộng được;

Hỗ trợ các tính năng ngôn ngữ lập trình hướng đối tượng, như thừa kế, interfaces, overloading...;

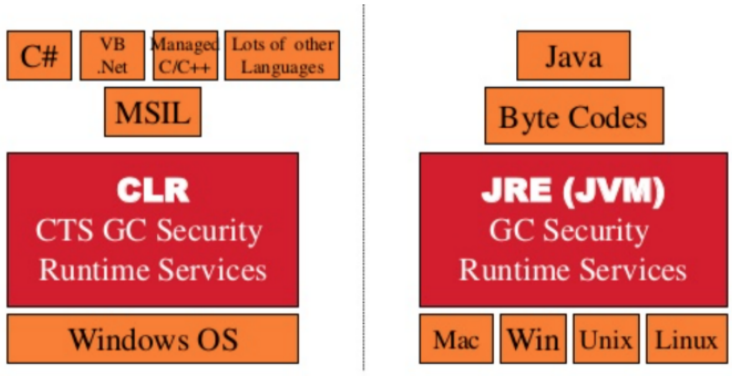
Hỗ trợ việc lập trình đa luồng (multi-threading);

Hỗ trợ bắt và xử lý ngoại lệ (exception handing);

Hỗ trợ các thuộc tính tùy biến;

Có cơ chế thu gom và giải phóng bộ nhớ;

Sử dụng delegates thay vì con trỏ hàm nhằm tăng tính bảo mật.



Hình 2.2 CLR và JRE ở 'trung gian' giữa ngôn ngữ và hệ điều hành [5]

2.3 Công cụ hỗ trợ

2.3.1 Wireshark

Wireshark là một công cụ phân tích giao thức mạng mạnh mẽ và miễn phí được sử dụng rộng rãi trong lĩnh vực bảo mật mạng, gỡ lỗi mạng, và phát triển ứng dụng mạng. Với khả năng ghi và phân tích gói tin mạng từ nhiều nguồn, Wireshark cung cấp một cái nhìn chi tiết và toàn diện về hoạt động mạng, giúp người dùng hiểu rõ cách hoạt động của mạng và tìm ra các vấn đề mạng cũng như các mối đe dọa bảo mật. Dưới đây là một số tính năng chính của Wireshark:

(1) Phân tích giao thức: Wireshark hỗ trợ hơn 2,000 giao thức mạng khác nhau, cho phép bạn phân tích và hiểu rõ cách hoạt động của các giao thức mạng như TCP, UDP, HTTP, DNS, SSL/TLS, v.v...

(2) Theo dõi và ghi lại gói tin: Wireshark cho phép bạn theo dõi và ghi lại gói tin mạng từ nhiều nguồn như card mạng, giao diện loopback, và các gói tin từ các máy chủ khác trên mạng.

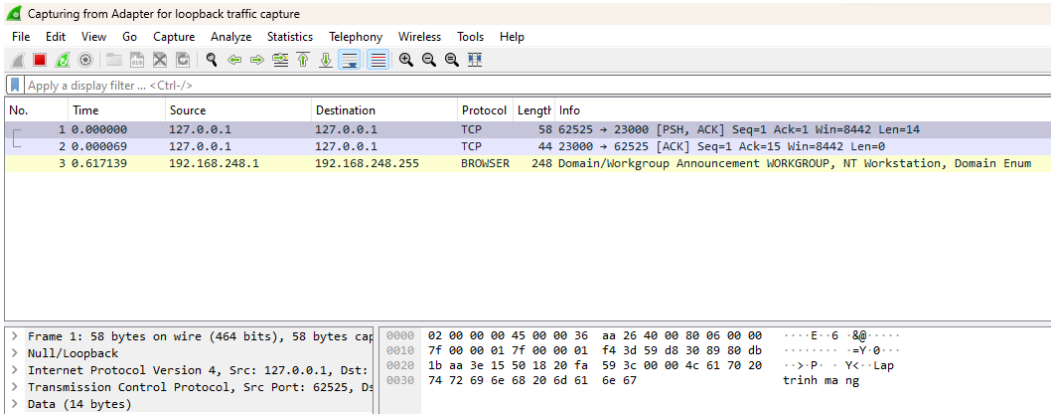
(3) Phân tích dữ liệu mạng: Bằng cách phân tích dữ liệu từ các gói tin mạng, Wireshark cung cấp các báo cáo và thống kê chi tiết về lưu lượng mạng, giao thức sử dụng, và thời gian phản hồi.

(4) Gỡ lỗi mạng: Wireshark cho phép bạn gỡ lỗi và tìm ra các vấn đề mạng như lỗi kết nối, gói tin bị mất, hoặc vấn đề về hiệu suất mạng.

(5) Kiểm tra bảo mật: Wireshark có thể giúp bạn kiểm tra và phát hiện các mối đe dọa bảo mật như tấn công DOS/DDoS, lỗ hổng bảo mật, và dữ liệu nhạy cảm được gửi không an toàn.

(6) Tích hợp với các công cụ khác: Wireshark có thể tích hợp với các công cụ và dịch vụ khác như TShark, tcpdump, Ngrep, v.v..., để cung cấp các tính năng mở rộng và mạnh mẽ hơn.

Wireshark là một công cụ quan trọng, không thể thiếu cho bất kỳ nhà phát triển mạng hoặc chuyên gia bảo mật nào, giúp họ phát hiện, phân tích, giải quyết các vấn đề mạng và bảo mật một cách hiệu quả.



Hình 2.3 Wireshark bắt gói tin TCP

2.3.2 Postman

Postman là một ứng dụng cung cấp một giao diện đồ họa thân thiện, dễ sử dụng để thử nghiệm, phát triển và tương tác với các API (Application Programming Interface). Nó cho phép người dùng tạo, gửi các yêu cầu HTTP, HTTPS, xem và phân tích các phản hồi từ server, quản lý biến môi trường, nhiều tính năng khác để hỗ trợ việc làm việc với API một cách hiệu quả. Dưới đây là một số tính năng chính của Postman:

(1) Tạo và gửi yêu cầu HTTP/HTTPS: Người dùng có thể dễ dàng tạo các yêu cầu HTTP và HTTPS như GET, POST, PUT, DELETE, v.v… gửi chúng đến các URL tương ứng.

(2) Quản lý các yêu cầu: Postman cho phép bạn tổ chức và quản lý các yêu cầu của mình vào các bộ sưu tập (collections), mỗi bộ sưu tập có thể chứa nhiều yêu cầu liên quan.

(3) Kiểm tra và phân tích phản hồi: Sau khi gửi yêu cầu, Postman hiển thị phản hồi từ server một cách rõ ràng và chi tiết, giúp bạn dễ dàng phân tích dữ liệu trả về.

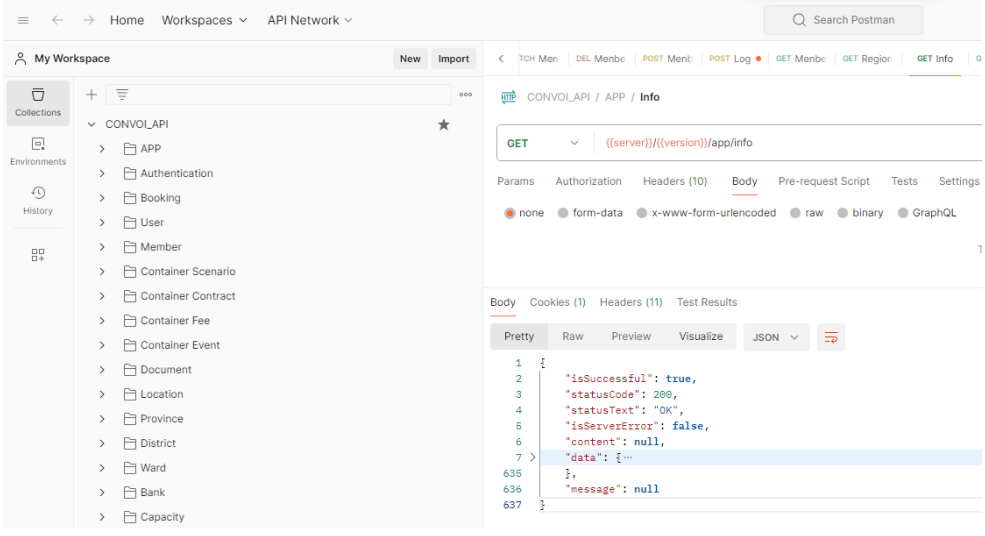
(4) Môi trường và biến: Postman cho phép bạn quản lý các môi trường (environments) và biến môi trường (environment variables), giúp bạn linh hoạt trong việc thử nghiệm các API trên nhiều môi trường khác nhau.

(5) Chia sẻ và hợp tác: Bạn có thể chia sẻ các bộ sưu tập của mình với đồng nghiệp hoặc cộng đồng Postman, cũng như làm việc cùng nhau trên cùng một dự án.

(6) Kiểm thử API: Postman cung cấp các tính năng kiểm thử và automation cho việc kiểm thử API, bao gồm việc tạo bộ kiểm thử (test suites), viết các kịch bản kiểm thử tự động (automated test scripts), tự động hóa các quy trình kiểm thử.

(7) Mở rộng và tích hợp: Postman có thể được mở rộng thông qua các add-ons và tích hợp với nhiều công cụ, dịch vụ khác như Newman, Jenkins, Slack, GitHub, v.v...

Postman là một công cụ mạnh mẽ và phổ biến trong cộng đồng phát triển phần mềm và API, giúp tăng hiệu suất, đồng bộ hóa quy trình làm việc với API.



Hình 2.4 Postman quản lý và gọi API

2.3.3 Fiddler

Fiddler là một công cụ gỡ lỗi và kiểm tra proxy mạng được sử dụng rộng rãi trong phát triển ứng dụng web. Nó cung cấp một giao diện đồ họa dễ sử dụng để theo dõi, ghi lại các yêu cầu HTTP/HTTPS giữa máy tính của bạn và máy chủ, cho phép bạn kiểm tra, phân tích các gói dữ liệu truyền qua mạng một cách chi tiết. Dưới đây là một số tính năng chính của Fiddler:

(1) Theo dõi yêu cầu và phản hồi HTTP/HTTPS: Fiddler cho phép bạn theo dõi và ghi lại tất cả các yêu cầu, phản hồi HTTP/HTTPS giữa máy tính của bạn và máy chủ, bao gồm cả các yêu cầu AJAX, các yêu cầu từ ứng dụng di động.

(2) Gỡ lỗi và kiểm tra hiệu suất: Bằng cách theo dõi và phân tích các yêu cầu, phản hồi mạng, Fiddler giúp bạn gỡ lỗi trong ứng dụng web của mình, kiểm tra hiệu suất để tối ưu hóa tốc độ tải trang.

(3) Chỉnh sửa yêu cầu và phản hồi: Fiddler cho phép bạn chỉnh sửa yêu cầu và phản hồi trực tiếp trong quá trình giao tiếp giữa máy tính của bạn và máy chủ, giúp bạn thử nghiệm các kịch bản, sửa đổi dữ liệu gửi và nhận.

(4) Kiểm tra bảo mật: Bằng cách theo dõi các yêu cầu và phản hồi mạng, Fiddler cho phép bạn kiểm tra bảo mật của ứng dụng bằng cách phát hiện các vấn đề như lỗ hổng bảo mật, dữ liệu nhạy cảm được gửi không an toàn, v.v...

(5) Tích hợp với các công cụ phát triển khác: Fiddler có thể tích hợp với các công cụ phát triển khác như Visual Studio, Chrome, Firefox, v.v..., để cung cấp một loạt các tính năng mạnh mẽ cho quá trình phát triển và gỡ lỗi ứng dụng web.

Fiddler là một công cụ mạnh mẽ, linh hoạt cho việc gỡ lỗi và kiểm tra ứng dụng web, cung cấp cho bạn khả năng theo dõi, kiểm soát hoàn toàn dữ liệu truyền qua mạng.

2.4 Thư viện và Frameworks

ASP.NET Core SignalR

SignalR là một thư viện mã nguồn mở giúp đơn giản hóa việc thêm chức năng web thời gian thực vào ứng dụng trong ASP.NET Core. Chức năng web thời gian thực cho phép mã nguồn phía máy chủ gửi nội dung tới máy khách ngay lập tức.

Các ứng dụng phù hợp với SignalR:

(1) Ứng dụng cần cập nhật tần suất cao từ máy chủ. Ví dụ như trò chơi, mạng xã hội, bình chọn, đấu giá, bản đồ và ứng dụng GPS.

(2) Ứng dụng bảng điều khiển và giám sát. Ví dụ như bảng điều khiển công ty, cập nhật bán hàng tức thì hoặc cảnh báo đi lại.

(3) Ứng dụng cộng tác. Ví dụ như ứng dụng bảng trắng và phần mềm họp nhóm.

(4) Ứng dụng yêu cầu thông báo. Ví dụ như mạng xã hội, email, trò chuyện, trò chơi, cảnh báo đi lại và nhiều ứng dụng khác.

SignalR cung cấp một API để tạo ra các cuộc gọi thủ tục từ máy chủ đến máy khách (RPC). Các RPC này kích hoạt các hàm trên máy khách từ mã nguồn phía máy chủ của .NET Core. Có một số nền tảng được hỗ trợ, mỗi nền tảng đi kèm với SDK khách hàng tương ứng. Do đó, ngôn ngữ lập trình được gọi bằng cuộc gọi RPC có thể thay đổi.

Dưới đây là một số tính năng của SignalR cho ASP.NET Core:

(1) Tự động quản lý kết nối.

(2) Gửi tin nhắn tới tất cả máy khách kết nối cùng một lúc. Ví dụ như một phòng trò chuyện.

(3) Gửi tin nhắn tới các máy khách cụ thể hoặc nhóm máy khách.

(4) Mở rộng để xử lý lưu lượng tăng lên.

Giao thức trung tâm của SignalR là SignalR Hub Protocol. SignalR hỗ trợ các kỹ thuật sau để xử lý giao tiếp thời gian thực (theo thứ tự của việc sa sút mềm mại):

(1) WebSockets;

(2) Server-Sent Events;

(3) Long Polling.

SignalR tự động chọn phương thức truyền tải tốt nhất trong khả năng của máy chủ và máy khách. SignalR sử dụng các hub để giao tiếp giữa máy khách và máy chủ.

Một hub là một ống dẫn cấp cao cho phép một máy khách và một máy chủ gọi các phương thức trên nhau. SignalR tự động xử lý việc chuyển giao qua các ranh giới máy, cho phép máy khách gọi các phương thức trên máy chủ và ngược lại. Bạn có thể truyền các tham số kiểu cố định cho các phương thức, điều này cho phép ràng buộc mô hình. SignalR cung cấp hai giao thức hub tích hợp sẵn: một giao thức văn bản dựa trên JSON và một giao thức nhị phân dựa trên MessagePack. MessagePack thường tạo ra các tin nhắn nhỏ hơn so với JSON. Các trình duyệt cũ phải hỗ trợ XHR level 2 để cung cấp hỗ trợ giao thức MessagePack.

Hubs gọi mã máy khách bằng cách gửi các tin nhắn chứa tên và tham số của phương thức máy khách. Các đối tượng được gửi như tham số phương thức được giải mã bằng giao thức đã được cấu hình. Máy khách cố gắng phù hợp tên với một phương thức trong mã máy khách. Khi máy khách tìm thấy một phù hợp, nó gọi phương thức và truyền cho nó dữ liệu tham số đã được giải mã.

Code

```
using Microsoft.AspNetCore.SignalR;

namespace SignalR_JavaScript.Hubs
{
    public class ChatHub:Hub
    {
        public async Task SendMessage(string user, string message)
        {
            await Clients.All.SendAsync("ReceiveMessage", user, message);
        }
    }
}
```

```
using SignalR_JavaScript.Hubs;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
```

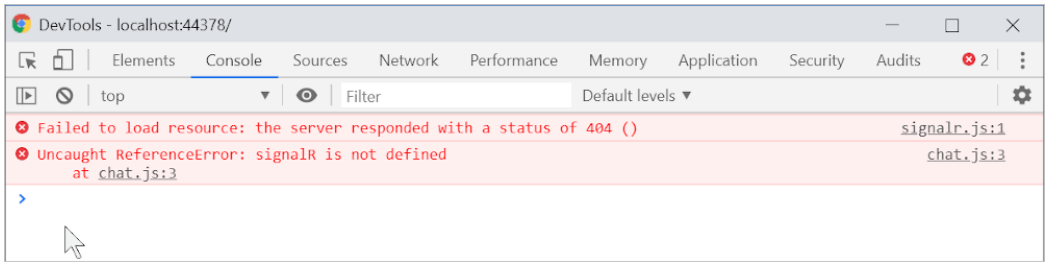
```
builder.Services.AddRazorPages();
builder.Services.AddSignalR();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Error");
    // The default HSTS value is 30 days. You may want to change this for production scenarios, see
https://aka.ms/aspnetcore-hsts.
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapRazorPages();
app.MapHub<ChatHub>("/chatHub");

app.Run();
```

```
@page
<div class="container">
    <div class="row p-1">
        <div class="col-1">User</div>
        <div   class="col-5"><input   type="text"   id="userInput"
/></div>
    </div>
    <div class="row p-1">
        <div class="col-1">Message</div>
        <div   class="col-5"><input   type="text"   class="w-100"
id="messageInput" /></div>
    </div>
    <div class="row p-1">
        <div class="col-6 text-end">
            <input   type="button"   id="sendButton"   value="Send
Message" />
        </div>
    </div>
    <div class="row p-1">
        <div class="col-6">
            <hr />
        </div>
    </div>
    <div class="row p-1">
        <div class="col-6">
            <ul id="messagesList"></ul>
        </div>
    </div>
</div>
<script src="~/js/signalr/dist/browser/signalr.js"></script>
```

```
<script src="~/js/chat.js"></script>
```

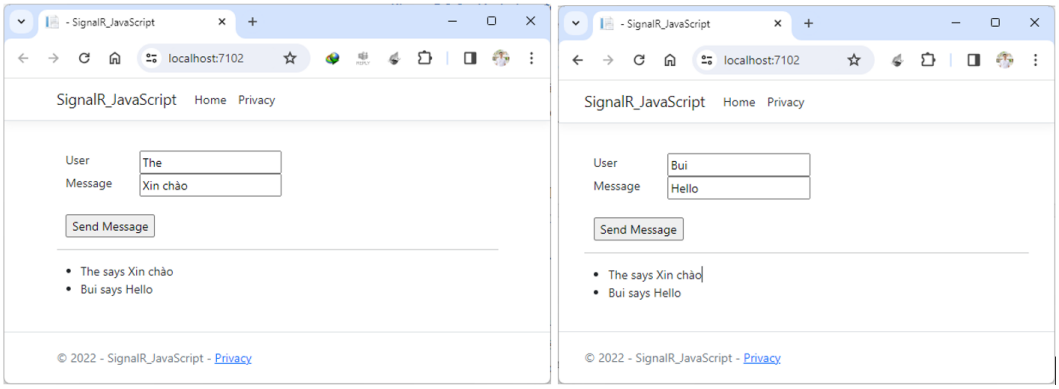
```
"use strict";

var connection = new signalR.HubConnectionBuilder().withUrl("/chatHub").build();

//Disable the send button until connection is established.
document.getElementById("sendButton").disabled = true;

connection.on("ReceiveMessage", function (user, message) {
    var li = document.createElement("li");
    document.getElementById("messagesList").appendChild(li);
    // We can assign user-supplied strings to an element's textContent because it
    // is not interpreted as markup. If you're assigning in any other way, you
    // should be aware of possible script injection concerns.
    li.textContent = `${user} says ${message}`;
});

connection.start().then(function () {
    document.getElementById("sendButton").disabled = false;
}).catch(function (err) {
    return console.error(err.toString());
});

document.getElementById("sendButton").addEventListener("click", function (event) {
    var user = document.getElementById("userInput").value;
    var message = document.getElementById("messageInput").value;
    connection.invoke("SendMessage", user, message).catch(function (err) {
        return console.error(err.toString());
    });
    event.preventDefault();
});
```

Nếu ứng dụng không hoạt động, hãy mở công cụ dành cho nhà phát triển trình duyệt (nhấn phím F12 trên trình duyệt) và đi tới bảng điều khiển. Tìm kiếm các lỗi có thể xảy ra liên quan đến mã HTML và JavaScript. Ví dụ: nếu signalr.js được đặt trong một thư mục khác với chỉ dẫn, tham chiếu đến tệp đó sẽ không hoạt động dẫn đến lỗi 404 trong bảng điều khiển.



Nếu ERR_SPDY_INADEQUATE_TRANSPORT_SECURITYxảy ra lỗi trong Chrome, hãy chạy các lệnh sau để cập nhật chứng chỉ phát triển:

```
dotnet dev-certs https --clean

dotnet dev-certs https --trust
```



Hình 2.5 Kết quả thử nghiệm với SignalR

HttpClient là một lớp quan trọng trong .NET Framework, cung cấp các phương thức linh hoạt để tạo ra và gửi các yêu cầu HTTP đến máy chủ, xử lý phản hồi từ máy chủ đó. Nó cung cấp một cách dễ dàng để tương tác với các dịch vụ web, API hoặc các nguồn dữ liệu khác trên Internet. Dưới đây là một ví dụ minh họa về cách sử dụng HttpClient để tải về nội dung của một trang web:

Code

```
using System;
using System.Net.Http;
using System.Threading.Tasks;

namespace HttpClientExample
{
   internal class Program
   {
      static async Task Main(string[] args)
      {
         // Khởi tạo HttpClient
         using (HttpClient client = new HttpClient())
         {
            try
            {
               // Gửi yêu cầu GET đến một URL cụ thể
               HttpResponseMessage response = await client.GetAsync("https://ut.edu.vn");

               // Kiểm tra xem yêu cầu có thành công không
               if (response.IsSuccessStatusCode)
               {
                  // Đọc nội dung phản hồi dưới dạng chuỗi
                  string content = await response.Content.ReadAsStringAsync();

                  // In ra nội dung của trang web đã tải về
                  Console.WriteLine(content);
                  Console.ReadKey();
               }
               else
               {
                  Console.WriteLine("Không thể tải về nội dung: " + response.StatusCode);
               }
            }
            catch (Exception ex)
```

```
            {
                Console.WriteLine("Lỗi: " + ex.Message);
            }
        }
    }
}
```

Trong ví dụ này, sử dụng HttpClient để gửi một yêu cầu GET đến URL "https://ut.edu.vn". Sau đó, kiểm tra xem yêu cầu có thành công không và nếu có, đọc nội dung của phản hồi dưới dạng chuỗi và in ra màn hình.

WebSocket: Thư viện hỗ trợ lập trình với giao thức WebSocket, giúp tạo ra kết nối hai chiều giữa client và server.

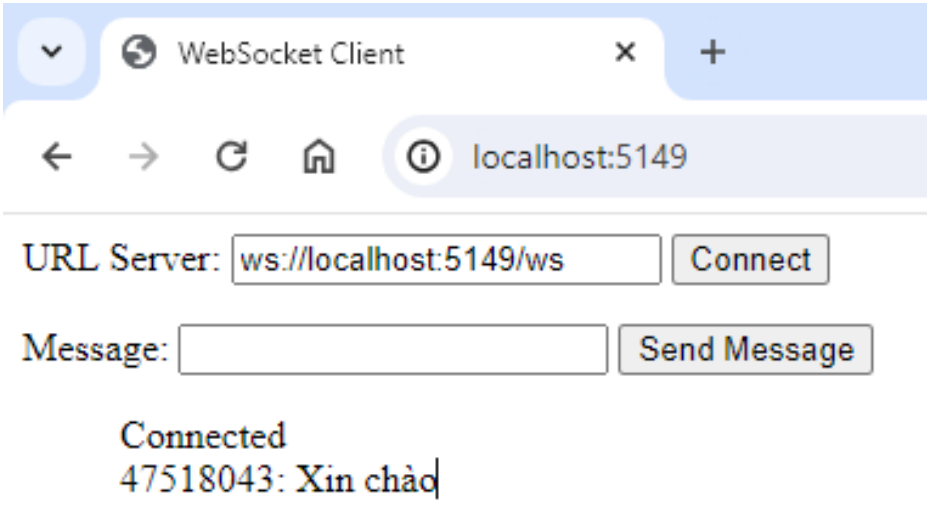
Code

```
/*Program.cs*/
using System.Net.WebSockets;
using System.Net;
using System.Text;

var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.UseWebSockets();

var websocket = new WebSocketOptions
{
    KeepAliveInterval = TimeSpan.FromSeconds(5),
};

app.MapGet("/", async context =>
{
    context.Response.ContentType = "text/html; charset=utf-8";

    var htmlContent = File.ReadAllText("./public/index.html");

    await context.Response.WriteAsync(htmlContent);
});

app.UseWebSockets();

var connectedClients = new List<WebSocket>();

app.Map("/ws", async context =>
{
    if (context.WebSockets.IsWebSocketRequest)
    {
        using var webSocket = await context.WebSockets.AcceptWebSocketAsync();
        connectedClients.Add(webSocket);
        int newClientHashCode = webSocket.GetHashCode();

        var rand = new Random();

        while (true)
        {
            var buffer = new byte[1024];
```

```
        var result = await webSocket.ReceiveAsync(new ArraySegment<byte>(buffer), CancellationToken.None);

        if (result.MessageType == WebSocketMessageType.Text)
        {
            var receivedMessage = Encoding.UTF8.GetString(buffer, 0, result.Count);

            await SendMessageToAllClients(receivedMessage, newClientHashCode);
        }

    }
  }
  else
  {
    context.Response.StatusCode = (int)HttpStatusCode.BadRequest;
  }
});

async Task SendMessageToAllClients(string message, int senderHashCode)
{
  foreach (var client in connectedClients)
  {
    if (client.State == WebSocketState.Open)
    {
      var messageWithSenderHashCode = $"{senderHashCode}: {message}";

      var messageData = Encoding.UTF8.GetBytes(messageWithSenderHashCode);

      await client.SendAsync(new ArraySegment<byte>(messageData), WebSocketMessageType.Text,
          true, CancellationToken.None);
    }
  }
}

app.Run();
```

```
<!--Index.html-->
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>WebSocket Client</title>

  <script type="text/javascript">
    let ws;

    function connect() {
      // Lấy giá trị từ input URL server
      let serverUrl = document.getElementById("serverUrl").value;

      let o = document.getElementById("output");

      // Kiểm tra xem URL có hợp lệ không
      if (serverUrl) {
        // Kết nối tới server
        ws = new WebSocket(serverUrl);
```

```
            o.innerText = "Connected";
            ws.onmessage = e => {
            console.log(e.data);
               generate(e.data);
            };

            ws.onclose = e => {
               o.innerText = e.reason;
            };
         } else {
            alert("Vui lòng nhập URL server trước khi kết nối.");
         }
      };

      function sendMessage() {
         let messageInput = document.getElementById("messageInput");
         let message = messageInput.value;

         // Kiểm tra xem có kết nối WebSocket hay không
         if (ws && ws.readyState === WebSocket.OPEN) {
            // Gửi tin nhắn tới server
            ws.send(message);
         } else {
            alert("Không có kết nối WebSocket hoặc kết nối đã đóng.");
         }

         // Xóa nội dung input
         messageInput.value = "";
      }

      function generate(text){
         var ul = document.getElementById("output");
         var li = document.createElement("li");
         li.appendChild(document.createTextNode(text));
         ul.appendChild(li);
      }
   </script>
   <style>
      #output {
         list-style-type: none;
      }
   </style>
</head>

<body>
   <label for="serverUrl">URL Server:</label>
   <input type="text" id="serverUrl" value="ws://localhost:5149/ws">
   <button onclick="connect()">Connect</button>
   <ul class="user-online" ></ul>

   <label for="messageInput">Message:</label>
   <input type="text" id="messageInput">
   <button onclick="sendMessage()">Send Message</button>

   <ul id="output"></ul>

</body>
</html>
```



Hình 2.6 Kết quả thử nghiệm WebSocket Client

SSL/TLS: Giới thiệu về bảo mật kết nối mạng và cách ứng dụng trong .NET. SSL (Secure Sockets Layer) và TLS (Transport Layer Security) là hai giao thức bảo mật được sử dụng để tạo kết nối an toàn giữa client và server trên mạng. Cả hai giao thức này đều cung cấp các tính năng bảo mật như mã hóa, xác thực và toàn vẹn dữ liệu, nhằm bảo vệ thông tin truyền qua mạng khỏi sự can thiệp của bên thứ ba.

Cách hoạt động của SSL/TLS

Handshake: Trong quá trình handshake, client và server thỏa thuận về các thông số bảo mật như phiên bản SSL/TLS, thuật toán mã hóa, và chứng chỉ số để xác thực lẫn nhau.

Xác thực: Server gửi chứng chỉ số của mình cho client để xác thực danh tính của mình. Client cũng có thể yêu cầu server gửi chứng chỉ số của CA (Certificate Authority) để xác thực chứng chỉ số của server.

Mã hóa: Sau khi quá trình handshake hoàn tất, client và server sử dụng các khóa mã hóa để mã hóa và giải mã dữ liệu truyền qua mạng.

Xác thực và toàn vẹn: SSL/TLS đảm bảo tính toàn vẹn, xác thực của dữ liệu truyền qua mạng bằng cách sử dụng hàm băm và mã hóa.

Ứng dụng trong .NET: .NET Framework cung cấp các lớp và thư viện để triển khai SSL/TLS trong ứng dụng của bạn. Cụ thể, có hai lớp quan trọng là SslStream và HttpClient.

SslStream: Lớp này cho phép bạn tạo một luồng an toàn để truyền dữ liệu qua mạng sử dụng SSL/TLS. Bạn có thể sử dụng SslStream để giao tiếp với các server HTTPS hoặc IMAP/SMTPs an toàn. Đây là một ví dụ cơ bản về cách sử dụng SslStream trong .NET:

Code

```
using System;
using System.IO;
using System.Net.Security;
using System.Net.Sockets;
using System.Text;

public class SSLClient
{
   public static void Main()
   {
      string serverName = "example.com";
      int port = 443;

      TcpClient client = new TcpClient(serverName, port);
      SslStream sslStream = new SslStream(client.GetStream());

      try
      {
         sslStream.AuthenticateAsClient(serverName);
         byte[] buffer = Encoding.UTF8.GetBytes("Hello, server!");
         sslStream.Write(buffer, 0, buffer.Length);
         buffer = new byte[4096];
         int bytesRead = sslStream.Read(buffer, 0, buffer.Length);
         Console.WriteLine(Encoding.UTF8.GetString(buffer, 0, bytesRead));
      }
      finally
      {
         sslStream.Close();
         client.Close();
      }
   }
}
```

HttpClient: Lớp này trong namespace System.Net.Http cho phép tạo và gửi các yêu cầu HTTP an toàn sử dụng SSL/TLS. Bạn có thể sử dụng HttpClient để tương tác với các API hoặc dịch vụ web yêu cầu kết nối an toàn. Dưới đây là một ví dụ về cách sử dụng HttpClient trong .NET:

Code

```
using System;
using System.Net.Http;
using System.Threading.Tasks;

public class SSLClient
{
   public static async Task Main()
   {
      string url = "https://api.example.com";
      HttpClient httpClient = new HttpClient();

      try
      {
         HttpResponseMessage response = await httpClient.GetAsync(url);
         string responseBody = await response.Content.ReadAsStringAsync();
         Console.WriteLine(responseBody);
      }
      finally
```

```
        {
            httpClient.Dispose();
        }
    }
}
```

Trong ví dụ trên, SSL/TLS sẽ được sử dụng tự động khi client kết nối với server thông qua SslStream hoặc HttpClient, giúp đảm bảo rằng dữ liệu truyền qua mạng là an toàn và bảo mật.

Dưới đây là một số nguyên tắc và thực hành tốt để bảo vệ ứng dụng mạng của bạn khỏi các mối đe dọa và tấn công:

(1) Xác thực: Sử dụng các biện pháp xác thực mạnh mẽ như mã thông báo (token), OAuth, JWT (JSON Web Tokens), hoặc các phương thức xác thực hai yếu tố (2FA) để đảm bảo rằng người dùng được xác thực chính xác trước khi truy cập vào ứng dụng của bạn.

(2) Phân quyền: Thực hiện phân quyền cẩn thận để đảm bảo rằng người dùng chỉ có quyền truy cập vào những tài nguyên mà họ cần và được ủy quyền.

(3) Bảo vệ dữ liệu: Sử dụng mã hóa để bảo vệ dữ liệu khi nó truyền qua mạng. Đảm bảo rằng mật khẩu và dữ liệu nhạy cảm được lưu trữ và truyền đi một cách an toàn.

(4) Kiểm soát đầu vào: Xác thực và kiểm tra đầu vào từ người dùng và các nguồn dữ liệu bên ngoài để đảm bảo rằng chúng không chứa mã độc hại hoặc có thể gây ra các loại tấn công như SQL Injection hay Cross-Site Scripting (XSS).

(5) Cập nhật hệ thống và phần mềm: Đảm bảo rằng hệ thống, phần mềm của bạn luôn được cập nhật với các bản và bảo mật mới nhất để bảo vệ khỏi các lỗ hổng bảo mật đã biết.

(6) Ghi nhật ký (Logging): Sử dụng ghi nhật ký để theo dõi các hoạt động trong hệ thống và phát hiện các hành vi bất thường hoặc tấn công đang diễn ra.

(7) Bảo vệ cơ sở hạ tầng mạng: Sử dụng tường lửa, giám sát mạng và các giải pháp bảo mật mạng để ngăn chặn các cuộc tấn công từ bên ngoài vào hệ thống của bạn.

(8) Tự học và cập nhật kiến thức về bảo mật: Tự học và cập nhật kiến thức về các mối đe dọa, kỹ thuật tấn công mới, cũng như các biện pháp bảo mật mới để giữ cho hệ thống của bạn luôn an toàn.

(9) Kiểm tra bảo mật định kỳ: Thực hiện kiểm tra bảo mật định kỳ và kiểm tra phát hiện xâm nhập để phát hiện và khắc phục các lỗ hổng bảo mật trong hệ thống của bạn.

(10) Hợp tác với cộng đồng bảo mật: Tham gia vào cộng đồng bảo mật để học hỏi từ những người khác và chia sẻ thông tin về các mối đe dọa, biện pháp bảo mật hiệu quả.

2.5 Tóm tắt và Kết luận

Chương 2 đã cung cấp cách nhìn tổng quan về các công cụ, môi trường phát triển và nguyên tắc quan trọng trong việc phát triển ứng dụng mạng trên .NET Framework. Qua chương này, chúng ta đã làm quen với các công cụ hỗ trợ như Wireshark, Postman và Fiddler, cũng như tìm hiểu về cách đảm bảo an toàn trong lập trình mạng với xác thực và mã hóa dữ liệu. Chương cũng giới thiệu về .NET Framework, các thư viện và frameworks hữu ích trong phát triển ứng dụng mạng. Điều này giúp chúng ta hiểu rõ hơn về cách lựa chọn công cụ và môi trường phát triển phù hợp để tạo ra các ứng dụng mạng hiệu quả, an toàn, đáng tin cậy. Đồng thời, chương cũng nhấn mạnh rằng sự lựa chọn đúng công cụ và môi trường là chìa khóa quan trọng để xây dựng các ứng dụng mạng thành công, có hiệu suất cao.

Câu hỏi ôn tập Chương 2

Câu 1. .NET Framework có hỗ trợ lập trình mạng không?

Câu 2. ASP.NET là gì và vai trò của nó trong .NET Framework?

Câu 3. WCF là gì và khi nào nên sử dụng trong lập trình mạng?

Câu 4. .NET Framework hỗ trợ giao thức nào cho việc giao tiếp giữa các ứng dụng mạng?

Câu 5. .NET Framework có hỗ trợ việc xử lý dữ liệu JSON không?

Câu 6. Làm thế nào để xây dựng một ứng dụng TCP Server trong .NET Framework?

Câu 7. .NET Framework hỗ trợ việc gửi email không?

Câu 8. Làm thế nào để tạo một ứng dụng web đơn giản trong .NET Framework?

Câu 9. .NET Framework hỗ trợ mã hóa dữ liệu không?

Câu 10. Làm thế nào để phát triển ứng dụng chat real-time trong .NET Framework?

Câu 11. Wireshark là gì và vai trò của nó trong lập trình mạng?

Câu 12. Postman là gì và những tính năng chính của nó trong lập trình mạng?

Câu 13. Fiddler là gì và tác dụng của nó trong lập trình mạng là gì?

Câu 14. Làm thế nào để sử dụng Wireshark để phân tích gói tin trong mạng?

Câu 15. Postman có thể sử dụng để thực hiện các loại yêu cầu HTTP nào?

Câu 16. Fiddler có thể giúp phát hiện và sửa lỗi gì trong ứng dụng mạng?

Câu 17. Làm thế nào để sử dụng Wireshark để kiểm tra giao tiếp giữa Client và Server?

Câu 18. Postman có thể tự động hóa quá trình kiểm thử API không?

Câu 19. Làm thế nào để sử dụng Fiddler để theo dõi lưu lượng mạng trên một ứng dụng cụ thể?

Câu 20. Các công cụ này có hỗ trợ mã hóa và xác thực dữ liệu trong giao tiếp mạng không?

Câu 21. C# là ngôn ngữ lập trình được sử dụng phổ biến trong lập trình mạng vì lý do gì?

Câu 22. NET Framework là gì và vai trò của nó trong lập trình mạng sử dụng C# là gì?

Câu 23. Các thành phần cơ bản của một ứng dụng mạng viết bằng C# là gì?

Câu 24. Tại sao Async và Await được sử dụng nhiều trong lập trình mạng với C#?

Câu 25. Làm thế nào để xử lý ngoại lệ trong môi trường lập trình mạng với C#?

Câu 26. Có những công cụ nào hỗ trợ phát triển ứng dụng mạng trong C#?

Câu 27. Làm thế nào để xử lý giao tiếp hai chiều trong một ứng dụng mạng viết bằng C#?

Câu 28. Làm thế nào để kiểm tra và giải quyết vấn đề đóng băng giao diện người dùng trong ứng dụng mạng với C#?

Câu 29. Làm thế nào để tối ưu hóa hiệu suất của ứng dụng mạng viết bằng C#?

Câu 30. Tại sao việc tránh đóng băng giao diện người dùng trong lập trình mạng với C# là quan trọng?