



# برنامه‌نویسی شبکه در C#

آشنایی با مفاهیم پایه و پیشرفته برنامه‌نویسی شبکه در زبان C#

مدرس: محمدحسین روحی‌امینی



 <https://www.linkedin.com/in/mhramini>

 [MohamadHoseinRoohiAmini](https://github.com/MohamadHoseinRoohiAmini)

الهام گرفته از Chris Woodruff: در سایت <https://csharp-networking.com>

# نقشه راه : مباحث برنامه‌نویسی شبکه در C#

مقدمه‌ای بر برنامه‌نویسی سوکت

مفاهیم بنیادی شبکه

بررسی اجمالی برنامه‌نویسی شبکه

استراتژی‌های مدیریت خطأ و تحمل خرابی

چندنخی در برنامه‌های شبکه

برنامه‌نویسی ناهمگام با Async/Await

کار با REST APIs

بهینه‌سازی عملکرد شبکه

تکنیک‌های سریالی‌سازی داده

کار با MQTT برای IoT

کار با WebRTC

کار با WebSockets

پیاده‌سازی صفحه‌ای پیام

کار با WebHooks

کار با gRPC

نگاهی به آینده با QUIC

استفاده از SignalR

# برنامه‌نویسی شبکه در C#

در این دوره جامع، اصول و تکنیک‌های برنامه‌نویسی شبکه در C# را به صورت کاربردی فرا خواهید گرفت. ما مفاهیم پیچیده شبکه را با روشی ساده و قابل فهم ارائه می‌دهیم تا بتوانید به سرعت مهارت‌های کاربردی را کسب کنید.

در طول دوره، با مفاهیم بنیادی شبکه، کلاس‌های System.Net، سوکت‌ها، ارتباطات TCP/IP و UDP آشنا می‌شوید. همچنین نحوه پیاده‌سازی برنامه‌های کلاینت-سرور، وب سرویس‌ها و API‌های RESTful را به صورت عملی خواهید آموخت.



# مقدمه‌ای بر برنامه‌نویسی شبکه

## تعریف و اهمیت

برنامه‌نویسی شبکه به فرآیند طراحی و ایجاد نرم‌افزارهایی اطلاق می‌شود که امکان تبادل داده بین سیستم‌های مختلف را در بستر شبکه‌های کامپیوتری فراهم می‌کنند. این حوزه کلیدی از توسعه نرم‌افزار می‌تواند در شبکه‌های محلی (LAN)، شبکه‌های گسترده جغرافیایی (WAN)، اینترنت جهانی و یا محیط‌های ترکیبی پیاده‌سازی شود، و نقشی اساسی در دنیای ارتباطات دیجیتال امروز ایفا می‌کند.

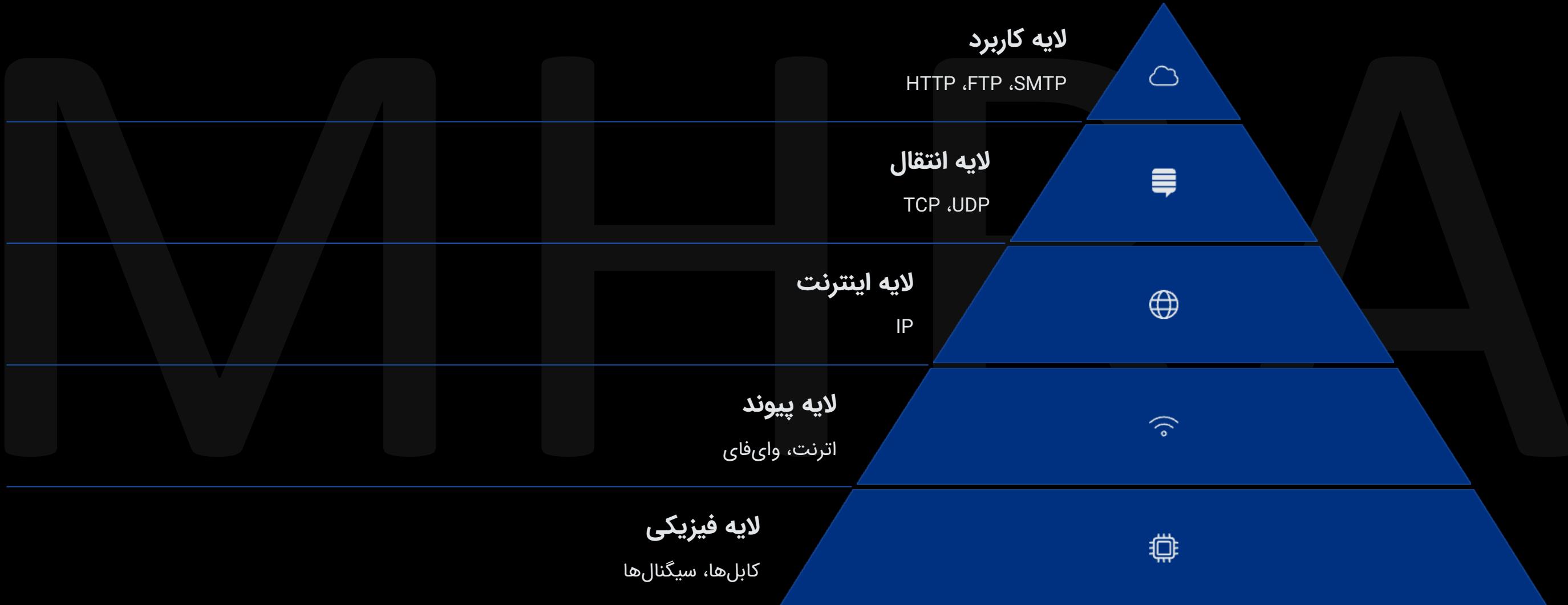
## مفاهیم کلیدی

سلط بر برنامه‌نویسی شبکه مستلزم درک عمیق چندین مفهوم بنیادی است. سوکت‌ها به عنوان نقاط پایانی ارتباط، دروازه‌هایی هستند که داده‌ها از طریق آنها جریان می‌یابند. آدرس‌های IP همچون آدرس‌های پستی دیجیتال، هویت منحصر به فرد هر دستگاه را در شبکه مشخص می‌کنند، در حالی که شماره‌های پورت، سرویس‌های خاص روی آن دستگاه را تعیین می‌نمایند. مکانیزم انتقال بسته نیز با تقسیم اطلاعات به قطعات کوچک‌تر، انتقال کارآمد و مطمئن داده‌ها را تضمین می‌کند.

## کاربردهای برنامه‌نویسی شبکه

دامنه کاربرد برنامه‌نویسی شبکه بسیار گسترده و متنوع است. معماری کلاینت-سرور، ستون فقرات بسیاری از سیستم‌های امروزی است که در آن کلاینت‌ها درخواست‌هایی را به سرورها ارسال کرده و پاسخ‌های مناسب دریافت می‌کنند. وب سرویس‌ها و API‌ها، پلتفرم‌های پیام‌رسانی و چت، بازی‌های آنلاین چندنفره، سیستم‌های رایانش ابری و اینترنت اشیا (IoT) همگی نمونه‌های بارز استفاده از این تکنولوژی حیاتی هستند.

# پروتکل‌های شبکه و ارتباطات



<https://www.linkedin.com/in/mohamad-hosein-roohi-amini/>

<https://github.com/MohamadHoseinRoohiAmini>

# معماری کلاینت-سرور



## تعامل

در این معماری، مسئولیت‌ها به‌طور شفاف تفکیک شده‌اند: کلاینت‌ها درخواست می‌دهند و سرورها خدمات ارائه می‌کنند. این جداسازی هوشمندانه، باعث افزایش مقیاس‌پذیری، امنیت بالاتر و مدیریت آسان‌تر سیستم‌های نرم‌افزاری می‌شود.



## سرور

سرورها قلب تپنده این معماری هستند. آنها دائمًا به درخواست‌ها گوش می‌دهند، آنها را با دقت پردازش کرده و پاسخ‌های مناسب و بهینه را به کلاینت‌ها ارسال می‌کنند.



## کلاینت

کلاینت‌ها آغازگر ارتباط هستند. آنها به سرورها متصل شده، درخواست‌های مشخص خود را ارسال می‌کنند و منتظر دریافت پاسخ می‌مانند.

# مدل درخواست-پاسخ



## فرستادن درخواست

درخواست از طریق پروتکل‌های شبکه مانند HTTP به سرور مقصد ارسال می‌شود

## پردازش درخواست

سرور درخواست را تحلیل کرده، عملیات مورد نیاز را انجام می‌دهد و نتایج را آماده می‌سازد

## شروع درخواست

کلاینت یک درخواست با پارامترها و داده‌های مشخص ایجاد می‌کند و آن را برای پردازش آماده می‌سازد

## فرستادن پاسخ

سرور پاسخ مناسب را با کد وضعیت و داده‌های درخواستی تولید کرده و به کلاینت بازمی‌گرداند

مدل درخواست-پاسخ یکی از اساسی‌ترین الگوهای تعامل در معماری وب و سیستم‌های توزیع شده است. در این مدل، کلاینت‌ها نیازهای خود را در قالب درخواست‌های ساختاریافته ارسال می‌کنند. سرورها پس از دریافت، این درخواست‌ها را پردازش کرده و نتایج مناسب را برمی‌گردانند. این الگوی ارتباطی در تمام فناوری‌های وب از مرور صفحات ساده تا سرویس‌های پیچیده API و تبادل داده‌های لحظه‌ای کاربرد گسترده‌ای دارد.

<https://www.linkedin.com/in/mohamadhosseini/>  
<https://github.com/MohamadHoseinRoohiAmini>

# مقیاس‌پذیری و متعادل‌سازی بار

## متعادل‌سازی بار

متعادل‌سازی بار تکنیکی است که درخواست‌های ورودی را به‌طور هوشمند بین چندین سرور توزیع می‌کند. این مکانیزم اطمینان می‌دهد که بار کاری به‌صورت متوازن تقسیم شده و از اضافه بار روی سرورهای منفرد جلوگیری می‌شود. نتیجه این رویکرد، پایداری بیشتر سیستم و زمان پاسخ‌دهی سریع‌تر برای کاربران است.

## افزایش کاربران

با افزایش محبوبیت برنامه‌ها، توانایی پشتیبانی از کاربران بیشتر ضروری می‌شود. مقیاس‌پذیری به معنای قابلیت سیستم برای رسیدگی به حجم فزاینده کاربران همزمان بدون افت کارایی است. این ویژگی کلیدی باید از مراحل اولیه معماری سیستم طراحی و پیاده‌سازی شود.

## پاسخگویی

حتی در شرایط اوج بار کاری و استفاده همزمان هزاران کاربر، سیستم باید پاسخگویی سریع و ثبات عملکرد خود را حفظ کند. طراحی مقیاس‌پذیر صحیح تضمین می‌کند که زمان پاسخ‌دهی حتی در لحظات پرترافیک همچنان در محدوده قابل قبول باقی بماند، و تجربه کاربری مطلوب همواره حفظ شود.

## بهینه‌سازی منابع

مدیریت کارآمد منابع سیستم (نظیر CPU، حافظه و پهنای باند (نقش حیاتی در عملکرد کلی دارد. با پیاده‌سازی استراتژی‌های مقیاس‌پذیری هوشمند، می‌توان بهره‌وری منابع را به حداقل رساند، هزینه‌های زیرساختی را کاهش داد و تأثیر زیستمحیطی کمتری بر جای گذاشت.

# اصول برنامه‌نویسی سوکت

## انواع سوکت

سوکت‌های TCP اتصال‌گرا هستند و با استفاده از مکانیسم‌های تأیید و ترتیب‌بندی، تحویل مطمئن داده‌ها را تضمین می‌کنند. در مقابل، سوکت‌های UDP بدون اتصال بوده و با حذف سربار پروتکل، انتقال سریع داده را در کاربردهای مانند پخش زنده و بازی‌های آنلاین که تأخیر کم اهمیت بیشتری از قابلیت اطمینان دارد، امکان‌پذیر می‌سازند.

## آدرس‌دهی سوکت

هر سوکت با ترکیبی منحصر به فرد از آدرس IP او شماره پورت شناسایی می‌شود. این ساختار آدرس‌دهی دوگانه (IP:Port) امکان ایجاد چندین اتصال همزمان روی یک دستگاه را فراهم می‌کند و مسیریابی دقیق بسته‌های داده در شبکه‌های پیچیده را تضمین می‌نماید.

## سوکت‌ها چیستند؟

سوکت‌ها رابطه‌های برنامه‌نویسی قدرتمندی هستند که ارتباط بین برنامه‌ها در شبکه را امکان‌پذیر می‌سازند. این ابزارها به عنوان نقاط پایانی ارتباطی عمل می‌کنند و چارچوبی استاندارد برای تبادل داده بین دستگاه‌های مختلف فراهم می‌کنند.

# برنامه‌نویسی شبکه در C# و .NET.

## </> C# 12 و .NET 8

این فناوری‌ها با پشتیبانی از مدل‌های پلتفرم‌های قدرتمندی برای توسعه برنامه‌های شبکه‌ای هستند .NET 8 و C# 12. برنامه‌نویسی مدرن و کتابخانه‌های غنی، امکان ساخت برنامه‌های شبکه‌ای مقیاس‌پذیر و کارآمد را فراهم می‌کنند.



### کتابخانه‌های شبکه

کتابخانه‌های System.Net.Sockets برای ارتباطات سطح پایین، System.Net.Http برای ابزارهای عمومی شبکه و System.Net.Sockets برای تعاملات HTTP طراحی شده‌اند. این مجموعه کامل، پیاده‌سازی پروتکل‌های متعدد شبکه را ساده می‌سازد.



### برنامه‌نویسی ناهمگام

الگوی `C#` در `async/await` امکان توسعه برنامه‌های چندنخی بدون پیچیدگی‌های سنتی را فراهم می‌کند. این روش عملکرد الارا حفظ کرده و با مدیریت بهینه منابع، کارایی برنامه‌های شبکه‌ای را به طور چشمگیری افزایش می‌دهد.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

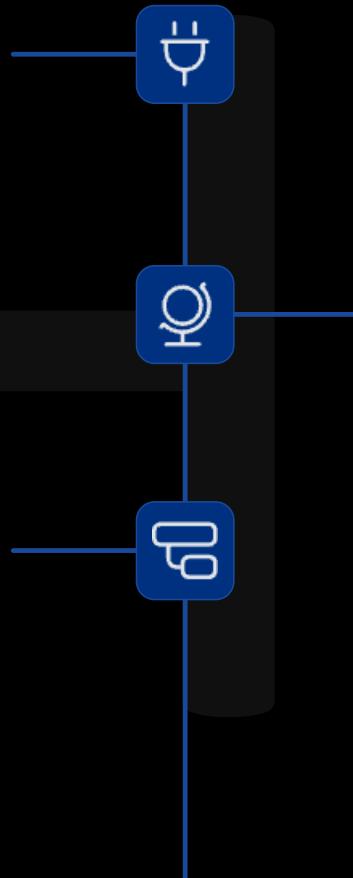
# کتابخانه‌های شبکه در .NET.

## System.Net.Sockets

این کتابخانه امکان برنامه‌نویسی سوکت سطح پایین را با پیاده‌سازی استاندارد Berkeley Socket فراهم می‌کند. با آن می‌توانید ارتباطات TCP/IP و UDP را پیاده‌سازی کنید و کنترل دقیقی روی بافرها، جریان‌های داده و تنظیمات شبکه داشته باشید.

## برنامه‌نویسی ناهمگام

مدل (TAP) Task-based Asynchronous Pattern، در C# با کلیدواژه‌های `async/await`، تعامل با شبکه را بدون مسدود کردن نخ اصلی ممکن می‌سازد. این روش باعث افزایش مقیاس‌پذیری و بهبود عملکرد در برنامه‌های با تعداد زیاد درخواست همزمان می‌شود.



## System.Net.Http

این کتابخانه قدرتمند API کاملی برای ارتباطات HTTP/HTTPS را ارائه می‌دهد HttpClient. بخش اصلی آن است که از الگوی مدرن کارخانه‌ای، سرآیندهای خودکار، فشرده‌سازی محتوا، احراز هویت و مدیریت کوکی‌ها پشتیبانی می‌کند.

این کتابخانه‌ها با استانداردهای مجموعه‌ای غنی از کتابخانه‌های شبکه ارائه می‌دهد که از پروتکل‌های سطح پایین تا سرویس‌های سطح بالا را پوشش می‌دهند.NET. با استفاده از این ابزارها می‌توانید برنامه‌های مقیاس‌پذیر و مدرن وب سازگار هستند و امنیت، کارایی و انعطاف‌پذیری را در توسعه برنامه‌های شبکه تضمین می‌کنند. چندسکویی توسعه دهید.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# فریمورک‌ها و کتابخانه‌های شبکه در .NET.



## SignalR

فریمورک قدرتمند برای ارتباطات بلادرنگ که انتقال داده‌ها را به صورت دوطرفه و آنی امکان‌پذیر می‌سازد. ایده‌آل برای توسعه اپلیکیشن‌های چت، بازی‌های آنلاین چندنفره و داشبوردهای نمایش اطلاعات زنده با کارایی بالا.



## gRPC

چارچوبی برای ارتباطات فوق العاده سریع و کم حجم بین سیستم‌های توزیع شده. با استفاده از پروتکل HTTP/2 و سریالی‌سازی کارآمد، عملکرد بهینه‌ای در ارتباطات میان میکروسرویس‌ها ارائه می‌دهد.



## MQTT

پروتکل سبک و کم مصرف که برای دستگاه‌های محدود اینترنت اشیاء بهینه‌سازی شده است. با حداقل مصرف پهنه‌ای باند در شبکه‌های نامطمئن و کندسرعت عملکرد عالی داشته و الگوی انتشار/اشتراک را برای ارتباطات پیاده‌سازی می‌کند.



## Azure SDK

مجموعه جامعی از کتابخانه‌ها و ابزارهای یکپارچه برای توسعه در پلتفرم ابری مایکروسافت. به برنامه‌نویسان امکان می‌دهد تا با کدنویسی ساده و روان، از خدمات پیشرفته ذخیره‌سازی، پردازش داده، هوش مصنوعی و خدمات شبکه در مقیاس جهانی بهره‌مند شوند.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# مفاهیم اساسی شبکه

## مبانی ارتباطات شبکه



شناخت عمیق چگونگی تعامل و ارتباط دستگاهها در محیط‌های دیجیتال مختلف

## پروتکل‌ها و استانداردها



تسلط بر قوانین، قراردادها و زبان‌های ارتباطی که امکان گفتگوی سازگار بین سیستم‌ها را فراهم می‌کنند

## ساختارهای شبکه



کاوش در انواع توپولوژی‌ها و معماری‌های شبکه که زیربنای تبادل داده‌ها را شکل می‌دهند

در عصر دیجیتال امروز، مفاهیم شبکه نقشی حیاتی در زیرساخت فناوری ایفا می‌کنند. شبکه‌ها فراتر از مسیرهای انتقال داده، بستری برای نوآوری، همکاری و تحول دیجیتال هستند. این سیستم‌های پیچیده، ستون فقرات ارتباطات مدرن را تشکیل می‌دهند و به دستگاه‌ها امکان می‌دهند محتوای چندرسانه‌ای را به

<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

# مفاهیم پایه شبکه‌های کامپیوتری

## آدرس دهنده IP و زیرشبکه‌سازی

سیستم‌هایی که برای شناسایی منحصر به فرد دستگاه‌ها در شبکه استفاده می‌شوند. آدرس IP همانند نشانی پستی دیجیتال، امکان ارسال و دریافت اطلاعات را فراهم می‌کند. زیرشبکه‌سازی به مهندسان اجازه می‌دهد شبکه‌های بزرگ را به بخش‌های مدیریت‌پذیر تقسیم کنند که باعث بهبود کارایی و امنیت می‌شود.

## مسیریابی و توپولوژی‌های شبکه

مسیریابی فرآیند تعیین بهینه‌ترین مسیر برای انتقال داده‌ها بین دستگاه‌های مختلف است. روترهای توپولوژی‌های الگوریتم‌های پیچیده، بسته‌های اطلاعاتی را به مقصد صحیح هدایت می‌کنند. توپولوژی‌های شبکه الگوهای ساختاری هستند که نحوه اتصال فیزیکی و منطقی دستگاه‌ها را تعریف می‌کنند، مانند ستاره‌ای، حلقوی یا مش.

## پروتکل‌های شبکه و ارتباطات

مجموعه استانداردهای رسمی که چارچوب ارتباط بین سیستم‌ها را تعیین می‌کنند. پروتکل‌ها مانند TCP/IP، HTTP و DHCP، قواعد تبادل داده، فرمتهای پیام، روش‌های رمزگذاری و مکانیزم‌های تصحیح خطای مشخص می‌کنند تا سیستم‌های ناهمگون بتوانند به طور مؤثر با یکدیگر تعامل داشته باشند.

## سرویس‌های شبکه و پورت‌ها

سرویس‌های شبکه برنامه‌های تخصصی هستند که عملکردهای خاصی مانند انتقال فایل، وب‌گردی یا ایمیل را ارائه می‌دهند. پورت‌ها نقاط پایانی منطقی هستند که با اعداد مشخص (مانند پورت 80 برای HTTP) تعریف می‌شوند و امکان برقراری چندین ارتباط همزمان روی یک دستگاه را فراهم می‌سازند، مشابه اتاق‌های مختلف در یک ساختمان.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# آدرس دهی IP و زیر شبکه سازی

## درگ زیر شبکه سازی

زیر شبکه سازی هنر تقسیم یک شبکه بزرگ به بخش های کوچک تر و قابل مدیریت تر است. این تکنیک به مدیران شبکه امکان می دهد منابع را بهینه تر تخصیص دهند، امنیت را به طور قابل توجهی افزایش دهند و گلوگاه های ترافیکی را کاهش دهند. زیر شبکه ها مانند محله هایی در یک شهر بزرگ عمل می کنند.

## نمادگذاری CIDR

یا مسیریابی بین دامنه ای بدون کلاس، انقلابی در مدیریت CIDR این روش با استفاده از ایجاد کرده است. آدرس های پیشوند های عددی ساده، امکان تخصیص انعطاف پذیر آدرس ها را فراهم می کند و باعث استفاده کارآمد تر از فضای محدود زبان استاندارد طراحان و CIDR امروزه می شود. آدرس های مدیران شبکه است.

## معرفی آدرس دهی IP

آدرس های IP اشناسه های عددی منحصر به فردی هستند که مانند آدرس پستی برای هر دستگاه در شبکه عمل می کنند. این آدرس ها به بسته های اطلاعاتی اجازه می دهند تا مسیر درست خود را در میان میلیون ها دستگاه متصل به اینترنت پیدا کنند. بدون آدرس IP، ارتباطات شبکه ای امکان پذیر نخواهد بود.

## ماسک های زیر شبکه

ماسک های زیر شبکه ابزارهای هوشمندی هستند که مرز بین بخش شبکه و بخش میزبان یک آدرس IP را مشخص می کنند. این مرزبندی برای هدایت دقیق داده ها بین شبکه های مختلف ضروری است و به روترهای کمک می کند تصمیمات مسیریابی دقیق تری بگیرند.

# زیر شبکه سازی و تکنیک های آن

## تقسیم فضای آدرس IP

زیر شبکه سازی امکان تقسیم بندی هوشمندانه آدرس های IP را فراهم می کند. این روش، مدیریت کارآمدتر منابع شبکه و بهینه سازی انتقال داده ها را ممکن می سازد. با تقسیم فضای وسیع آدرس دهی به بخش های کوچکتر و منطقی، سازماندهی و نظارت بر شبکه به طور قابل توجهی تسهیل می شود.

## مزایای امنیتی

زیر شبکه سازی با ایجاد مرزبندی های منطقی، دستگاه ها را به گروه های مجزا تفکیک می کند. این جداسازی، انتشار تهدیدات امنیتی را محدود ساخته و نفوذ به کل شبکه را دشوارتر می کند. مدیران می توانند سیاست های امنیتی سفارشی را برای هر زیر شبکه پیاده سازی کرده و لایه های دفاعی قدر تمندتری ایجاد نمایند.

## کاهش ترافیک پخش همگانی

پیام های پخش همگانی (Broadcast) تنها در محدوده زیر شبکه مربوطه منتشر می شوند، نه در سراسر شبکه. این محدود سازی، بار ترافیکی را به طور چشمگیری کاهش داده و کارایی انتقال داده ها را افزایش می دهد. در محیط های شبکه ای پیچیده با تعداد زیاد دستگاه، این ویژگی به بهبود قابل توجه عملکرد شبکه منجر می شود.

## حفظ آدرس های IP

با توجه به محدودیت آدرس های IPv4، زیر شبکه سازی راه کاری موثر برای استفاده بهینه از این منابع ارزشمند است. سازمان ها می توانند با ایجاد زیر شبکه های متعدد در یک شبکه اصلی، تخصیص آدرس ها را بهینه سازی کرده و از اتلاف آنها جلوگیری کنند. این رویکرد امکان مدیریت کارآمدتر فضای آدرس دهی را فراهم می آورد.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# مسیریابی و شبکهای شبکه

## شکل حلقه‌ای

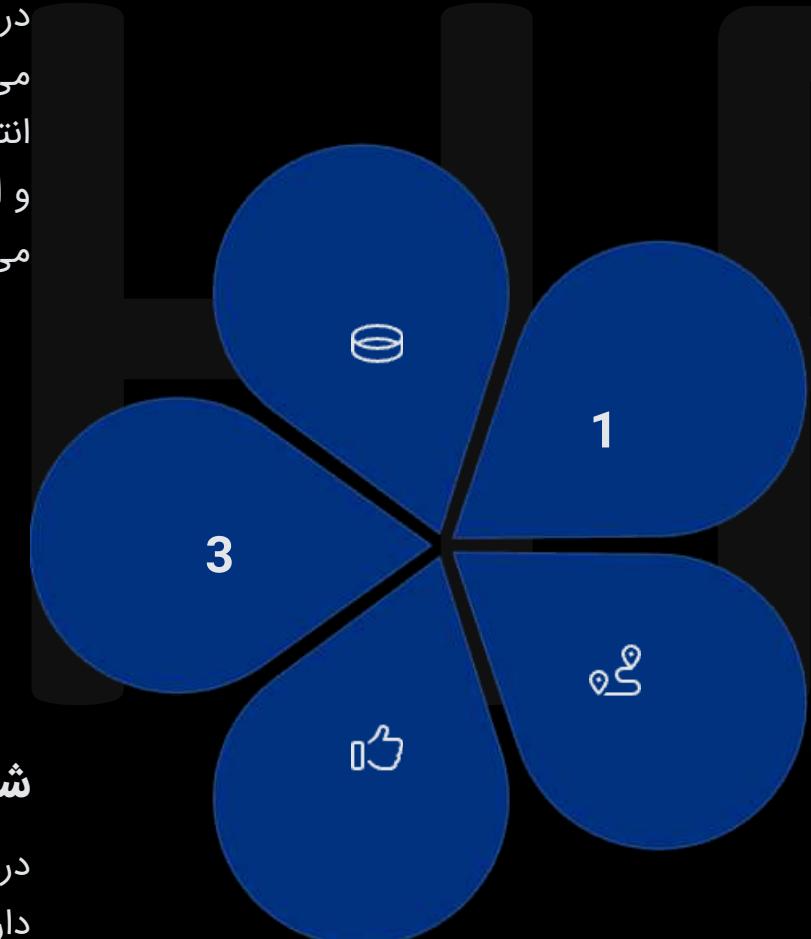
در توپولوژی حلقه‌ای، هر دستگاه دقیقاً به دو همسایه خود متصل می‌شود و داده‌ها در یک مسیر مشخص (ساعتگرد یا پاد ساعتگرد) انتقال می‌یابند. این ساختار قابلیت اطمینان بالایی در انتقال داده دارد و از ازدحام ترافیک جلوگیری می‌کند، اما قطع شدن حتی یک اتصال می‌تواند یکپارچگی کل شبکه را به خطر بیندازد.

## شکل درختی

این توپولوژی سلسله‌مراتبی، از یک دستگاه ریشه شروع می‌شود که به چندین دستگاه در لایه‌های پایین‌تر متصل است. ساختار درختی برای شبکه‌های گستره‌های سازمانی بسیار کارآمد است و امکان سازماندهی منطقی، مدیریت بهینه منابع و گسترش آسان شبکه را فراهم می‌سازد.

## شکل تور مانند

در توپولوژی مش، هر دستگاه با چندین دستگاه دیگر ارتباط مستقیم دارد که مسیرهای متعددی برای انتقال داده ایجاد می‌کند. این ساختار تحمل‌پذیری خطای بالایی دارد زیرا حتی با قطع شدن چندین اتصال، مسیرهای جایگزین همچنان در دسترس هستند. توپولوژی مش برای محیط‌های حیاتی که نیازمند پایداری و امنیت بالا هستند مانند



## شکل ستاره‌ای

در این توپولوژی، تمام دستگاه‌ها به یک نقطه مرکزی (مانند سوئیچ، روتر یا هاب) متصل می‌شوند. این ساختار مدیریت مرکزی و عیب‌یابی سریع را فراهم می‌کند، اما نقطه شکست واحد محسوب می‌شود؛ یعنی اگر مرکز از کار بیفتد، ارتباط کل شبکه مختل می‌شود.

## مسیریابی هوشمند

الگوریتم‌های مسیریابی پیشرفته، به صورت دینامیک بهینه‌ترین مسیر برای انتقال بسته‌های داده را بر اساس معیارهایی چون تأخیر، پهنای باند و بار ترافیکی شناسایی می‌کنند. این سیستم‌های هوشمند قادرند در زمان واقعی به تغییرات توپولوژی شبکه، خرابی تجهیزات یا افزایش ناگهانی ترافیک واکنش نشان داده و مسیرها را به طور خودکار بازپیکربندی کنند.

[زیرساخت‌های نظامی، ایده‌آل است.](https://www.linkedin.com/in/nbramini/)

<https://github.com/MohamadHoseinRoohiAmini>

# پروتکل‌های شبکه و ارتباطات

## لایه کاربرد



پروتکل‌هایی مانند HTTP، SMTP و FTP که رابط بین نرم‌افزارهای کاربردی و شبکه را فراهم می‌کنند و خدمات مستقیم به کاربران ارائه می‌دهند.

## لایه انتقال



بدون اتصال ثابت برای ارتباطات سریع‌تر با حجم کمتر استفاده UDP با کنترل جریان و تأیید دریافت برای انتقال مطمئن داده‌ها، و TCP می‌شود.

## لایه اینترنت



پروتکل IP که مسئول آدرس‌دهی منطقی، مسیریابی بسته‌های اطلاعاتی و تضمین رسیدن داده‌ها به مقصد صحیح در سراسر شبکه‌های مختلف است.

## لایه دسترسی به شبکه



پروتکل‌های پایه‌ای مانند اترنت، وای‌فای و بلوتوث که ارتباط فیزیکی بین دستگاه‌ها را مدیریت کرده و سیگنال‌های الکترونیکی را به داده‌های دیجیتال تبدیل می‌کنند.

معماری لایه‌ای شبکه همچون یک ساختار مهندسی دقیق عمل می‌کند. مدل‌های استانداردی مانند OSI چهارلایه، ارتباطات پیچیده را به وظایف مشخص و مدیریت‌پذیر تقسیم می‌کنند. هر لایه با استفاده از پروتکل‌های تخصصی، وظایف حیاتی مانند بسته‌بندی داده‌ها، آدرس‌دهی، کنترل خط‌خواه و امنیت را به صورت مستقل اما هماهنگ انجام می‌دهد.

<https://www.linkedin.com/in/mohamad-hosein-roohi-amini/>

<https://github.com/MohamadHoseinRoohiAmini>

# پروتکل‌های شبکه و عملکرد آنها

## بسته‌بندی داده‌ها

پروتکل‌های شبکه داده‌ها را به بسته‌های کوچک‌تر تقسیم می‌کنند تا انتقال کارآمدتر شود. این فرآیند امکان مدیریت بهینه جریان اطلاعات را فراهم می‌کند و در صورت از دست رفتن قسمتی از داده‌ها، بازیابی آنها را تسهیل می‌نماید.



## افزودن هدر

هر لایه پروتکل، اطلاعات کنترلی ضروری را به صورت هدر به بسته‌ها می‌افزاید. این هدرها حاوی آدرس‌های مبدأ و مقصد، کدهای تشخیص خط، شماره توالی و سایر متادیتای لازم برای تضمین انتقال دقیق و مطمئن هستند.



## مسیریابی

پس از آماده‌سازی بسته‌ها، پروتکل‌های مسیریابی بهترین مسیر برای رساندن آنها به مقصد را تعیین می‌کنند. این تصمیم‌گیری هوشمند بر اساس معیارهایی مانند تراکم شبکه، فاصله توپولوژیکی، پهنای باند موجود و تأخیر انجام می‌شود.



## بازسازی

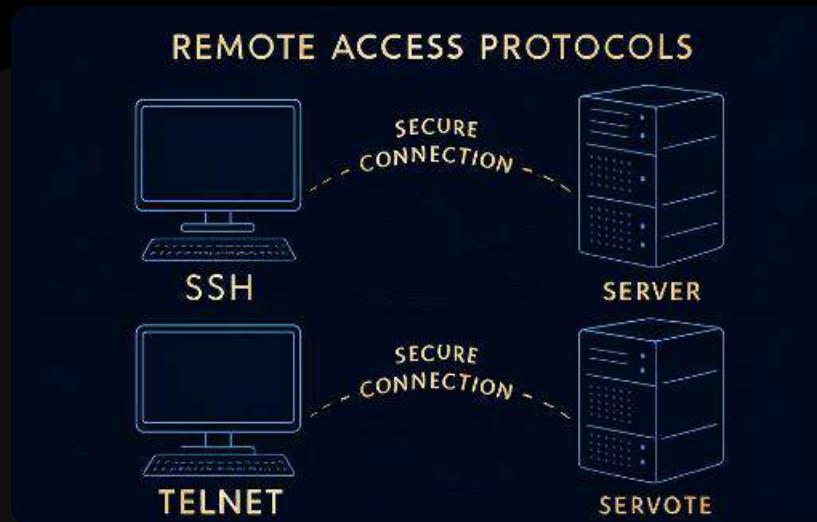
در سیستم مقصد، بسته‌های دریافتی به دقت مجدداً سرهمندی می‌شوند تا داده‌های اصلی بازیابی شود. پروتکل‌های لایه‌ای در گیرنده، به ترتیب هدرها را پردازش کرده و داده‌های خالص را برای استفاده در لایه‌های بالاتر و برنامه‌های کاربردی آماده می‌سازند.



<https://www.linkedin.com/in/mhramini/>

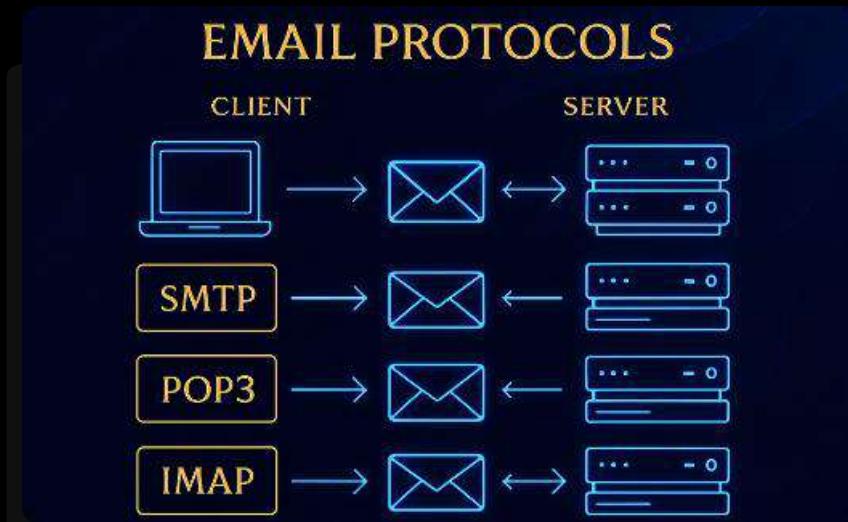
<https://github.com/MohamadHoseinRoohiAmini>

# سرویس‌های شبکه و پورت‌ها



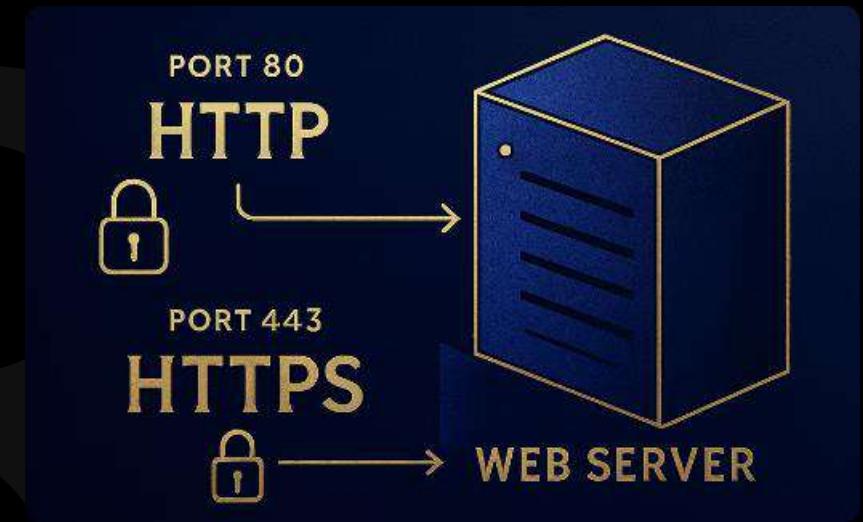
## سرویس‌های دسترسی از راه دور

مدیریت سرورها و سیستم‌ها از راه دور با استفاده از پورت SSH (22 یا ) پورت Telnet (23) انجام می‌شود SSH . با رمزگذاری قوی، ارتباطی امن ایجاد می‌کند و برای محیط‌های حرفه‌ای و تجاری توصیه می‌شود Telnet . اطلاعات را به صورت متن ساده بدون رمزگذاری (منتقل می‌کند و استفاده از آن در شبکه‌های عمومی می‌تواند خطر سرقت اطلاعات را



## سرویس‌های ایمیل

پروتکل SMTP پورت 25 مسئول ارسال ایمیل‌ها بین سرورهاست . برای دریافت ایمیل‌ها، کاربران می‌توانند از POP3 پورت 110 که ایمیل‌ها را دانلود می‌کند یا IMAP پورت 143 که امكان مدیریت ایمیل روی سرور را فراهم می‌سازد، استفاده کنند IMAP . برای کاربرانی که به ایمیل‌های خود از چندین دستگاه دسترسی دارند، مناسب‌تر است.



## سرویس‌های وب

وبسایت‌ها از پروتکل HTTP پورت 80 برای انتقال محتوا و HTTPS پورت 443 برای ارتباطات امن استفاده می‌کنند . هنگام مرور اینترنت، مرورگر شما خودکار با این پورت‌ها ارتباط برقرار می‌کند . HTTPS نسخه‌ای رمزگذاری شده از HTTP است که با استفاده از گواهینامه‌های SSL/TLS، از اطلاعات شخصی و تراکنش‌های مالی شما محافظت می‌کند.

# اسکن پورت و شناسایی سرویس‌ها

80

HTTP

پورت استاندارد برای انتقال محتوای وب که اکثر وب‌سایت‌ها از آن برای ارسال صفحات، تصاویر و داده‌های غیررمزنگاری شده استفاده می‌کنند

443

HTTPS

نسخه امن HTTP که با استفاده از پروتکل‌های رمزنگاری SSL/TLS، ارتباط امن بین مرورگر و سرور را تضمین می‌کند

22

SSH

پروتکل امن برای مدیریت از راه دور سیستم‌ها که تمام ترافیک را رمزنگاری کرده و جایگزین امنی برای Telnet است

53

DNS

سیستم نام‌گذاری دامنه که نام‌های دامنه قابل خواندن برای انسان را به آدرس‌های IP قابل استفاده برای کامپیوترها ترجمه می‌کند

اسکن پورت و شناسایی سرویس، ابزارهای حیاتی در زرادخانه هر متخصص امنیت و مدیر شبکه هستند. اسکن پورت فرآیند سیستماتیک بررسی یک سیستم یا شبکه برای یافتن پورت‌های باز، بسته یا فیلترشده است. شناسایی سرویس یک گام فراتر رفته و مشخص می‌کند چه نرم‌افزار و نسخه‌ای روی هر پورت در حال اجراست. این اطلاعات برای آشکارسازی آسیب‌پذیری‌های امنیتی، بهینه‌سازی عملکرد شبکه و اطمینان از انطباق با سیاست‌های امنیتی سازمان ضروری است.

<https://www.linkedin.com/in/mohamad-hosein-roohi-a mini/>  
<https://github.com/MohamadHoseinRoohiAmini>

# خلاصه و آینده برنامه‌نویسی شبکه



این دوره به ما امکان داد تا با عمق و گستردگی دنیای برنامه‌نویسی شبکه آشنا شویم. ما نه تنها مفاهیم بنیادی شبکه را فرا گرفتیم، بلکه با ساختارهای پیچیده، اصطلاحات تخصصی و پروتکل‌های متنوع آن به طور کامل آشنا شدیم. در طول این مسیر، موضوعات کلیدی و استراتژیک مانند آدرس دهی IP، تکنیک‌های زیرشبکه‌سازی، الگوریتم‌های مسیریابی و طبقه‌بندی انواع شبکه را به صورت عمیق بررسی و تحلیل کردیم.

در افق آینده، برنامه‌نویسی شبکه با ظهور و گسترش فناوری‌های نوین مانند شبکه‌های 5G، اکوسیستم اینترنت اشیاء، محاسبات ابری پیشرفته و سیستم‌های هوش مصنوعی دستخوش تحولات شگرفی خواهد شد. برنامه‌نویسان C# با تسلط بر اصول بنیادی و همگام‌سازی مداوم با کتابخانه‌های نوین .NET، قادر خواهند بود نه تنها با چالش‌های پیچیده آینده به طور مؤثر روبرو شوند، بلکه از فرصت‌های بی‌نظیر این عرصه به منظور خلق راهکارهای نوآورانه و کارآمد بهره‌مند شوند.



# برنامه‌نویسی سوکت

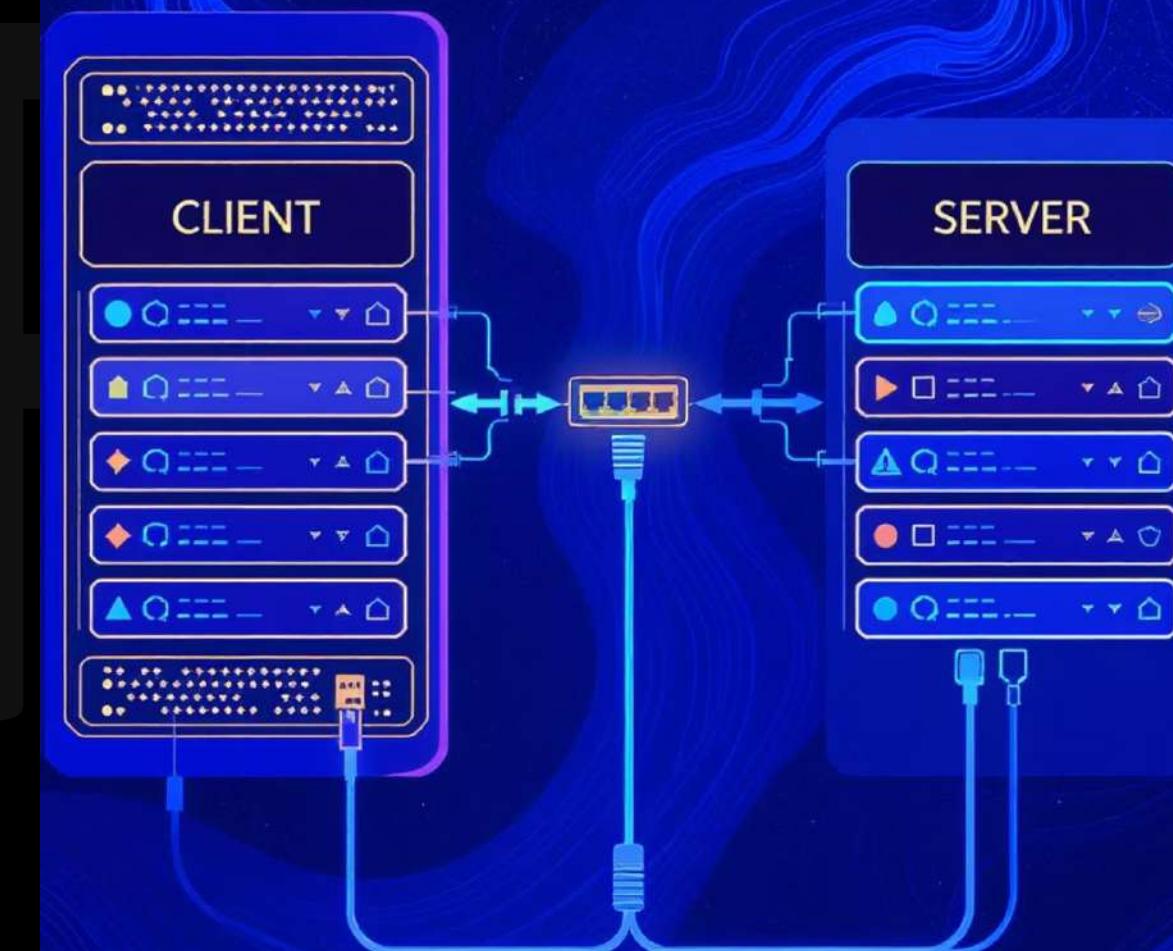
آشنایی با مفاهیم پایه و پیشرفته برنامه‌نویسی شبکه در C#

# CLIENT-SERVER

## مقدمه‌ای بر برنامه‌نویسی سوکت

در دنیای گسترده شبکه‌های کامپیوتری، جایی که اطلاعات مانند رودی دیجیتال جریان دارد، برنامه‌نویسی سوکت پلی اساسی است که دستگاه‌ها، برنامه‌ها و کاربران را به هم متصل می‌کند. این فصل سفری را برای کشف هنر و علم برنامه‌نویسی سوکت آغاز می‌کند -مهارتی ضروری برای هر توسعه‌دهنده‌ای که در پیچیدگی‌های ارتباطات شبکه‌ای حرکت می‌کند.

برنامه‌نویسی سوکت روشی است که به برنامه‌های نرم‌افزاری اجازه می‌دهد کانال‌های ارتباطی، معروف به سوکت‌ها، را برای تبادل داده در سراسر شبکه ایجاد کنند.



<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# موضوعات اصلی

## مرور کلی برنامه‌نویسی سوکت

آشنایی با مفاهیم پایه، پروتکل‌ها و ساختارهای اصلی در برنامه‌نویسی سوکت

## اهمیت برنامه‌نویسی سوکت

بررسی چرایی اهمیت برنامه‌نویسی سوکت در دنیای مدرن شبکه و نقش آن در ارتباطات

## برنامه‌نویسی سوکت سمت سرور

تکنیک‌های ایجاد سرور، مدیریت چندین کلاینت و پردازش درخواست‌ها

## برنامه‌نویسی سوکت سمت کلاینت

چگونگی ایجاد، اتصال و مدیریت سوکت‌ها از دیدگاه کلاینت

# اهمیت برنامه‌نویسی سوکت

در عصر دیجیتال، ارتباط بین کامپیوترها، دستگاهها و برنامه‌های نرم‌افزاری یک ضرورت اساسی است. همانطور که انسان‌ها از طریق زبان‌ها و روش‌های مختلف ارتباط برقرار می‌کنند، کامپیوترها نیز به رویکردی ساختارمند برای انتقال داده‌ها به یکدیگر نیاز دارند. برنامه‌نویسی سوکت -سنگ پنای دنیای شبکه‌های کامپیوترا- این تبادل پیچیده داده را امکان‌پذیر می‌سازد.

برنامه‌نویسی سوکت ستون فقرات بسیاری از تعاملات دیجیتالی است که امروزه بدیهی می‌پنداریم. چه در حال مرور وب‌سایت مورد علاقه خود باشد، چه در حال کنفرانس ویدیویی آنلاین، یا انتقال فایل بین دستگاه‌ها، سوکت‌ها در پشت صحنه مشغول کار هستند.



<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# کاربردهای برنامه‌نویسی سوکت

## ایمیل

کلاینت‌های ایمیل از سوکت‌ها برای اتصال به سرورهای ایمیل استفاده می‌کنند و پیام‌ها را به صورت یکپارچه در سراسر اینترنت ارسال و دریافت می‌کنند.

## سرвис‌های وب

هنگام مرور وب، برنامه‌نویسی سوکت در پشت صحنه فعال است، اتصال به سرورهای وب را برقرار می‌کند، صفحات وب را دریافت می‌کند و محتوا را به مرورگر شما تحویل می‌دهد.

## ارتباطات بلادرنگ

سوکت‌ها برنامه‌های چت بلادرنگ، پلتفرم‌های کنفرانس ویدیویی و بازی‌های آنلاین را قدرت می‌بخشند و امکان تبادل فوری داده‌ها را فراهم می‌کنند.

## انتقال فایل

پروتکل‌هایی مانند (FTP) انتقال فایل (از سوکت‌ها برای انتقال فایل‌ها بین دستگاه‌ها استفاده می‌کنند).

# نقش سوکت‌ها

برای درک واقعی جوهره برنامه‌نویسی سوکت، ابتدا باید نقش محوری سوکت‌ها در هماهنگی ارتباطات شبکه را درک کنیم. در اصل، یک سوکت به عنوان نقطه پایانی در یک کانال ارتباطی عمل می‌کند و به عنوان دروازه‌ای است که از طریق آن داده‌ها می‌توانند بین دو موجودیت در یک شبکه ارسال و دریافت شوند.

سوکت‌ها را به عنوان درگاه‌های دیجیتالی تصور کنید که در آن پیام‌ها (داده‌ها) پهلو می‌گیرند، ارسال می‌شوند یا دریافت می‌شوند و گفتگویی دوطرفه بین برنامه‌های نرم‌افزاری را تسهیل می‌کنند.



# انواع سوکت

2

## سوکت‌های داده‌نگار(سوکت‌های UDP)

سوکت‌های داده‌نگار از پروتکل داده‌نگار کاربر (UDP) برای ارتباط استفاده می‌کنند. آنها بدون اتصال هستند، به این معنی که بسته‌های داده (داده‌نگارها) (بدون برقراری اتصال اختصاصی به صورت جداگانه ارسال می‌شوند.

- سرعت UDP: معمولاً سریع‌تر از TCP عمل می‌کند
- بدون تأیید: بسته‌ها ممکن است گم شوند یا خارج از ترتیب دریافت شوند
- سبک: به دلیل عدم وجود فرآیندهای برقراری و قطع اتصال

1

## سوکت‌های جریانی(سوکت‌های TCP)

سوکت‌های جریانی از پروتکل کنترل انتقال (TCP) برای ارتباط استفاده می‌کنند. آنها اتصال محور هستند و قبل از هرگونه انتقال داده، یک اتصال پایدار برقرار می‌کنند.

- قابلیت اطمینان TCP: تحويل بسته‌ها را تضمین می‌کند
- مرتب‌سازی: بسته‌های داده به ترتیب ارسال دریافت می‌شوند
- دوطرفه: امکان انتقال داده دوطرفه را فراهم می‌کند

# انواع سوکت (ادامه)

2

## سوکت‌های بسته ترتیبی

این سوکت‌ها ترکیبی از سوکت‌های جریانی و داده‌نگار هستند. آنها از خدمات اتصال‌محور استفاده می‌کنند اما داده‌ها را در رکوردها یا بسته‌های مشخص نگه می‌دارند.

- تحويل قابل اعتماد :مانند TCP، تحويل بسته را تضمین می‌کند
- حفظ مرزها :برخلاف TCP، مرزهای بسته را حفظ می‌کند
- موارد استفاده :انتقال داده‌های مبتنی بر رکورد یا زمانی که هم قابلیت اطمینان و هم حفظ مرز داده مورد نیاز است

1

## سوکت‌های خام

سوکت‌های خام دسترسی مستقیم‌تری به پروتکل‌های ارتباطی زیربنایی فراهم می‌کنند و به توسعه‌دهندگان امکان می‌دهند بسته‌های سفارشی ایجاد کنند یا پروتکلی را پیاده‌سازی کنند که به طور بومی توسط سیستم پشتیبانی نمی‌شود.

- سفارشی‌سازی :کنترل دقیق بر ایجاد و پردازش بسته ارائه می‌دهد
- مستقل از پروتکل :می‌تواند با هر پروتکل انتقال یا شبکه استفاده شود
- استفاده پیشرفته :به دلیل کنترل سطح پایین‌تر، به دانش عمیق‌تر پروتکل‌های شبکه نیاز دارد

# مرور کلی برنامه‌نویسی سوکت

در شبکه‌های کامپیوتری، جایی که دستگاه‌های سراسر جهان باید به طور یکپارچه ارتباط برقرار کنند، برنامه‌نویسی سوکت به عنوان محور اصلی ظاهر می‌شود که این باله پیچیده تبادل داده را هماهنگ می‌کند. در این بخش، سفری را برای رمزگشایی برنامه‌نویسی سوکت آغاز می‌کنیم و درک سطح بالایی از مفاهیم و اجزای اصلی آن ارائه می‌دهیم.

## پورت‌ها

در کنار آدرس‌های IP، پورت‌ها به تعریف بیشتر کانال‌های ارتباطی کمک می‌کنند. در حالی که یک آدرس IP مشابه آدرس یک ساختمان است، یک پورت مشابه یک آپارتمان در داخل آن ساختمان است.

## آدرس‌های IP

هر دستگاه متصل به شبکه دارای شناسه منحصر به فردی به نام آدرس IP است. این آدرس نقش مهمی در اطمینان از رسیدن بسته‌های داده به مقصد مورد نظر دارد.

## پروتکل‌ها

ارتباط در شبکه‌ها توسط قوانین یا پروتکل‌های استاندارد شده اداره می‌شود. دو پروتکل رایج در برنامه‌نویسی سوکت TCP و UDP هستند. هر کدام مزایا و موارد استفاده متمایزی دارند.

# ایجاد و پیکربندی سوکت

برای برقراری ارتباط دستگاه‌ها از طریق شبکه، سوکت‌ها باید ایجاد شوند. این شامل ایجاد این نقاط پایانی ارتباطی و پیکربندی آنها است، مشابه راهاندازی خطوط تلفن برای یک مکالمه. در برنامه‌نویسی سوکت، API‌ها ابزار لازم برای این کار را فراهم می‌کنند.

## ایجاد سوکت در C#

اولین قدم در ایجاد سوکت در C# شامل نمونه‌سازی یک شیء از کلاس System.Net.Sockets است. این کلاس در فضای نام Socket قرار دارد.

```
Socket newSocket = new Socket(AddressFamily.InterNetwork,  
SocketType.Stream, ProtocolType.Tcp);
```

در این مثال، سوکت برای یک آدرس IPv4 (به عنوان یک سوکت جریانی) معمولاً (AddressFamily.InterNetwork) با TCP استفاده می‌شود (ایجاد شده و پروتکل TCP را مشخص می‌کند).

## تنظیم گزینه‌های سوکت

پس از ایجاد سوکت، گزینه‌های مختلفی را می‌توان برای تنظیم رفتار آن پیکربندی کرد. این کار با استفاده از متدهای SetSocketOption انجام می‌شود.

```
socket.SetSocketOption(SocketOptionLevel.Socket,  
SocketOptionName.SendBuffer, 8192);
```

```
socket.SetSocketOption(SocketOptionLevel.Socket,  
SocketOptionName.ReceiveBuffer, 8192);
```

این تنظیمات اندازه باف ارسال و دریافت را به 8 کیلوبایت تنظیم می‌کنند.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# آدرس دهی سوکت

در دنیای دیجیتال، درست مانند دنیای فیزیکی، برای ارسال چیزی به کسی به یک آدرس نیاز دارید. سوکت‌ها نیز متفاوت نیستند. ترکیبی از یک آدرس IP و یک شماره پورت به طور منحصر به فرد هر سوکت را شناسایی می‌کند. آدرس IP دستگاه را در شبکه مشخص می‌کند و شماره پورت یک سرویس خاص را روی آن دستگاه شناسایی می‌کند.

## شماره‌های پورت ویژه

لازم به ذکر است که در حالی که محدوده شماره پورت از ۰ تا ۶۵۵۳۵ است، برخی محدوده‌ها اهمیت ویژه‌ای دارند:

- پورت‌های شناخته شده: (0-1023) برای سرویس‌های استاندارد مانند HTTP پورت (80) و FTP پورت (21) رزرو شده‌اند.
- پورت‌های ثبت شده: (1024-49151) معمولاً توسط برنامه‌های نرم‌افزاری استفاده می‌شوند.
- پورت‌های پویا/خصوصی: (49152-65535) می‌توانند آزادانه توسط نرم‌افزار بدون نیاز به ثبت استفاده شوند.

## اصول آدرس دهی سوکت

یک آدرس سوکت به عنوان یک شناسه منحصر به فرد عمل می‌کند که مشخص می‌کند داده‌ها باید به کجا ارسال یا از کجا دریافت شوند.

این آدرس ترکیبی از موارد زیر است:

- آدرس IP: انشان دهنده هویت یک ماشین در شبکه است.
- شماره پورت: یک عدد 16 بیتی که یک فرآیند یا برنامه خاص را روی ماشین شناسایی می‌کند.

# حالات ارتباطی سوکت

1

## حالت مسدودکننده

در حالت مسدودکننده، یک عملیات سوکت (مانند ارسال یا دریافت داده (اجرای برنامه را تا زمان تکمیل متوقف می‌کند. این حالت پیش‌فرض برای سوکت‌ها در .NET است.

- مزایا: برنامه‌نویسی را ساده می‌کند زیرا عملیات‌ها ساده و ترتیبی هستند
- معایب: می‌تواند باعث شود برنامه‌ها غیرپاسخگو باشند، به خصوص اگر عملیات شبکه زمان زیادی طول بکشد

2

## حالت غیرمسدودکننده

در حالت غیرمسدودکننده، عملیات‌های سوکت فوراً برمی‌گردند، حتی اگر وظیفه مورد نظر را تکمیل نکرده باشند. برنامه باید وضعیت را بررسی کند یا از مکانیزم‌های دیگر برای اطمینان از تکمیل استفاده کند.

- مزایا: امکان پاسخگویی برنامه‌ها را فراهم می‌کند زیرا با عملیات‌های طولانی شبکه متوقف نمی‌شوند
- معایب: نیاز به الگوهای برنامه‌نویسی پیچیده‌تر مانند نظرسنجی یا استفاده از انتخاب‌کننده‌ها دارد

# حالت‌های ارتباطی سوکت (ادامه)

2

## حالت همزمان

عملیات‌های همزمان عملیاتی هستند که در آن برنامه منتظر می‌ماند تا وظیفه سوکت قبل از ادامه کامل شود. در حالی که مشابه حالت مسدودکننده است، تمرکز در اینجا بر توالی عملیات‌ها است نه ماهیت مسدودکننده.

- مزایا: جریان عملیات را ساده می‌کند و برای مبتدیان آسان‌تر است
- معایب: مانند حالت مسدودکننده، می‌تواند برنامه‌ها را در طول وظایف طولانی غیرپاسخگو کند

1

## حالت ناهمzman

عملیات‌های ناهمzman به برنامه اجازه می‌دهند وظایف سوکت را آغاز کند که در پس زمینه اجرا می‌شوند و به رشتہ اصلی برنامه اجازه می‌دهد عملیات خود را ادامه دهد. پس از تکمیل وظیفه، یک متد فراخوانی بازگشتی فراخوانی می‌شود.

- مزایا: پاسخگویی حالت غیرمسدودکننده را با الگوهای برنامه‌نویسی شهودی‌تر ترکیب می‌کند. به ویژه برای برنامه‌های سرور مقیاس‌پذیر مناسب است
- معایب: ممکن است برای مبتدیان منحنی یادگیری بیشتری داشته باشد

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# برنامه‌نویسی سوکت سمت کلاینت

در تار و پود برنامه‌نویسی سوکت، تمایز واضحی بین دو بازیگر اصلی وجود دارد: سرور و کلاینت. در حالی که سرورها اغلب مسئول مدیریت و گوش دادن به اتصالات ورودی هستند، کلاینت‌ها نقشی به همان اندازه محوری ایفا می‌کنند. سمت کلاینت برنامه‌نویسی سوکت شامل مجموعه‌ای از روش‌ها و قراردادهایی است که تعیین می‌کند چگونه برنامه‌ها، به عنوان کلاینت، اتصالات به سرورها را آغاز، مدیریت و بسته می‌کند.

## اتصال به سرور

کلاینت به دنبال یک سرور می‌گردد و درخواست برقراری اتصال می‌کند.

## ایجاد سوکت

کلاینت یک سوکت ایجاد می‌کند که با نیازهای ارتباطی از نظر پروتکل و نوع داده مطابقت دارد.

## بستن اتصال

پس از اتمام کار، کلاینت اتصال را به طور مناسب می‌بندد.

## تبدیل داده

پس از برقراری اتصال، کلاینت می‌تواند داده‌ها را ارسال و دریافت کند.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# مدل کلاینت-سرور

مدل کلاینت-سرور یک مفهوم اساسی در محاسبات شبکه است و به عنوان ستون فقرات بیشتر برنامه‌های آنلاین امروزی، از مرور وب گرفته تا بازی‌های آنلاین، عمل می‌کند. در اصل، این مدل وظایف محاسباتی را بین دو موجودیت اصلی تقسیم می‌کند: سرورها، که مجموعه‌ای از خدمات را ارائه می‌دهند، و کلاینت‌ها، که این خدمات را درخواست می‌کنند. تعامل آنها اساس طیف گسترده‌ای از ارتباطات و تراکنش‌های دیجیتال را تشکیل می‌دهد.

## سرورها

سرورها ماشین‌ها یا برنامه‌های نرم‌افزاری قدرتمندی هستند که به درخواست‌های ورودی از کلاینت‌ها گوش می‌دهند. نقش اصلی آنها ارائه خدمات است، خواه ارائه یک صفحه وب، پخش یک ویدیو یا مدیریت بازی‌های چندنفره آنلاین باشد. یک سرور می‌تواند همزمان به چندین کلاینت خدمات ارائه دهد و درخواست هر کلاینت را به صورت منظم و کارآمد مدیریت کند.

## کلاینت‌ها

کلاینت‌ها، از سوی دیگر، آغازکنندگان این رابطه هستند. آنها می‌توانند هر چیزی از یک مرورگر وب روی لپتاپ، یک برنامه موبایل روی گوشی هوشمند یا یک برنامه نرم‌افزاری سفارشی روی یک ایستگاه کاری باشند. کلاینت‌ها برای دسترسی به خدمات یا منابع خاص به سرورها مراجعه می‌کنند.

# ایجاد سوکت و اتصال

ایجاد سوکت و اتصال گام‌های اساسی در مسیر برنامه‌نویسی سوکت سمت کلاینت است که برنامه، به عنوان یک کلاینت، یک سوکت ایجاد می‌کند و از آن برای ارتباط با یک سرور استفاده می‌کند. درک این فرآیند بسیار مهم است، زیرا لحن تمام تعاملات بعدی بین کلاینت و سرور را تعیین می‌کند.



## تپادل داده

پس از برقراری اتصال، کلاینت می‌تواند شروع به ارتباط با سرور کند، ارسال درخواست‌ها و دریافت پاسخ‌ها. این فرآیند شامل تبدیل داده‌ها به بایت‌ها و ارسال آنها از طریق سوکت است.



## اتصال به سرور

با ایجاد یک سوکت، گام بعدی اتصال به یک سرور است. برای این کار، کلاینت باید آدرس IP‌سرور و شماره پورتی که سرور در حال گوش دادن است را بداند. برای نمایش این اطلاعات، C# کلاس `IPEndPoint` ارائه می‌دهد.



## ایجاد سوکت

در C#, با استفاده از .NET، کلاس `Socket` موجود در فضای نام `System.Net.Sockets` ابزار اصلی برای ایجاد و مدیریت سوکت‌ها است. یک نمونه سوکت جدید را می‌توان با ارائه سه اطلاعات کلیدی ایجاد کرد: خانواده آدرس، نوع سوکت و نوع پروتکل.

# ارسال داده

برقراری اتصال بین یک کلاینت و یک سرور صحنه را برای مهمترین جنبه برنامه‌نویسی سوکت سمت کلاینت آماده می‌کند: "تبادل داده". ارسال داده "شامل روش‌ها و ظرافت‌های نحوه ارسال اطلاعات توسط کلاینت به سرور است. اما این روش نیاز به مدیریت دقیق دارد تا یکپارچگی، کارایی و قابلیت اطمینان داده‌ها تضمین شود.

## انتقال داده با استفاده از سوکت

هنگامی که داده‌ها در قالب بایت آماده شدند، متدهای `Send` از کلاس `Socket` به کار می‌آید. این متدهای بایت را می‌گیرد و آن را از طریق شبکه به سرور متصل ارسال می‌کند.

مثال کد:

```
int bytesSent = clientSocket.Send(byteData);
```

متدهای `Send` یک عدد صحیح را برمی‌گرداند که نشان‌دهنده تعداد بایت‌هایی است که با موفقیت ارسال شده‌اند.

## ارسال داده در بایت‌ها

در اصل، سوکت‌ها با بایت‌های خام سروکار دارند. چه در حال ارسال یک پیام متنی ساده باشید یا یک شیء سریالی شده پیچیده، داده‌ها باید قبل از انتقال به بایت تبدیل شوند. ابزارهای مختلفی برای تسهیل این تبدیل ارائه می‌دهد.

مثال کد:

```
string dataToSend = "Hello, server!";
byte[] byteData = Encoding.ASCII.GetBytes(dataToSend);
```

# دریافت داده

در هر مکالمه‌ای، گوش دادن به اندازه صحبت کردن مهم است. ارسال داده در ارتباط کلاینت-سرور حیاتی است، دریافت داده نیمه دیگر معادله است. پس از اینکه یک کلاینت اتصال برقرار می‌کند و درخواستی ارسال می‌کند، اغلب منتظر پاسخی از سرور است. این می‌تواند یک تأییدیه، بخشی از اطلاعات درخواست شده یا هر داده دیگری باشد.

## تبدیل بایت‌های دریافت شده

پس از دریافت داده‌ها در قالب بایت، اغلب نیاز به تبدیل آن به قالبی قابل استفاده، مانند یک رشته، دارد. با استفاده از کلاس Encoding، این تبدیل ساده است:

مثال کد:

```
string receivedMessage = Encoding.UTF8.GetString(buffer,  
0, bytesReceived);  
  
Console.WriteLine($"Message from server:  
{receivedMessage}");
```

## اصول دریافت داده

در C#، روش اصلی برای دریافت داده توسط یک سوکت کلاینت، متدهای Receive است. این متدهای آرایه بایت را با داده‌های ارسال شده توسط سرور پر می‌کند.

مثال کد:

```
byte[] buffer = new byte[1024];  
  
int bytesReceived = clientSocket.Receive(buffer);
```

متغیر bytesReceived می‌دهد چند بایت در بافر خوانده شده است. این اطلاعات مفید است، به خصوص اگر اندازه بافر بزرگتر از داده‌های واقعی دریافت شده باشد.

# مدیریت خطا و خاموشی مناسب

یکی از ویژگی‌های برنامه‌نویسی سوکت سمت کلاینت قوی، چگونگی برخورد مؤثر با خطاها احتمالی و تضمین خاموشی مناسب است. درست مانند هر شکل دیگری از ارتباط، ارتباط مبتنی بر سوکت مستعد وقفه‌ها و ناهنجاری‌ها است. در دنیای برنامه‌های شبکه‌ای، خطاها مدیریت نشده می‌توانند منجر به فساد داده‌ها، خرابی برنامه و تجربه‌های کاربری ضعیف شوند.

2

## مدیریت خطای اساسی

در C#, بلوک try-catch یک ساختار اساسی برای مدیریت استثناهای است. در برنامه‌نویسی سوکت، قرار دادن عملیات‌های سوکت در این بلوک‌ها می‌تواند از خرابی‌های غیرمنتظره جلوگیری کند:

```
try {  
    clientSocket.Connect(endPoint);  
}  
  
catch (SocketException se) {  
  
    Console.WriteLine($"Socket error: {se.Message}");  
}  
  
catch (Exception e) {  
  
    Console.WriteLine($"Unexpected error: {e.Message}");  
}
```

1

## تشخیص خطاهای احتمالی

برنامه‌نویسی سوکت می‌تواند با انواع مختلفی از خطاها مواجه شود، از جمله:

- اختلالات شبکه
- عدم دسترسی یا خاموشی سرور
- مدت زمان تایم‌اوت بیش از حد
- مسائل مربوط به رمزگذاری و رمزگشایی داده‌ها

هر یک از این موقعیت‌ها می‌تواند استثنایی را پرتاپ کند که در صورت عدم مدیریت، می‌تواند برنامه را متوقف کند.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# برنامه‌نویسی سوکت سمت سرور

برنامه‌نویسی سوکت سمت کلاینت خود در طرح کلی ارتباطات شبکه‌ای قرار دارد. در قلمرو گسترده برنامه‌های متصل، در حالی که کلاینت‌ها به عنوان جستجوگران خدمات یا داده‌ها عمل می‌کنند، سرورها نقش محوری ارائه‌دهندگان را ایفا می‌کنند. چه ارائه یک صفحه وب، مدیریت ترافیک ایمیل یا انتقال فایل‌ها باشد، پشت هر یک از این وظایف یک سرور است که با دقت به اتصالات ورودی گوش می‌دهد و درخواست‌ها را برآورده می‌کند.

## درباره و انتقال داده

سرورها درخواست‌های متنوعی را دریافت می‌کنند، از بازیابی داده‌ها گرفته تا انجام عملیات. بسته به این درخواست‌ها، سرورها داده‌های مورد نیاز را بازیابی و ارسال می‌کنند یا تکمیل وظایف را تأیید می‌کنند.

## مدیریت همزمانی

برخلاف یک کلاینت که معمولاً اتصال خود را مدیریت می‌کند، سرورها اغلب چندین اتصال را به طور همزمان مدیریت می‌کنند. این نیاز به مکانیزم‌های مدیریت همزمانی کارآمد دارد تا اطمینان حاصل شود که همه کلاینت‌ها پاسخ‌های به موقع دریافت می‌کنند.

## گوش دادن به اتصالات

سرورها دائمًا منتظر اتصالات ورودی کلاینت هستند. هنگامی که یک کلاینت به دنبال برقراری اتصال است، سرور درخواست را ارزیابی می‌کند و بر اساس پیکربندی‌ها و سیاست‌های خود، آن را می‌پذیرد یا رد می‌کند.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# ایجاد یک سوکت سرور

اساس برنامه‌نویسی سوکت سمت سرور، ایجاد یک سوکت سرور است. این موجودیت به عنوان یک دروازه خوشامدگویی عمل می‌کند که به طور مداوم به درخواست‌های اتصال کلاینت ورودی گوش می‌دهد. ساخت این دروازه به طور کارآمد و مؤثر برای اطمینان از ارتباط بدون درز، به حداقل رساندن تأخیرها و هموار کردن راه برای عملیات بعدی ضروری است.



## گوش دادن به اتصالات ورودی

پس از اتصال، سوکت سرور وارد حالت گوش دادن می‌شود و منتظر درخواست‌های اتصال ورودی می‌ماند. متدهای Listen و Accept این کار را انجام می‌دهند و پارامتری را می‌پذیرد که حداقل تعداد درخواست‌های اتصال در حال انتظار در صف را تعیین می‌کند.



## اتصال سوکت

اتصال، سوکت را با یک نقطه پایانی خاص مرتبط می‌کند که شامل یک آدرس IP و یک شماره پورت است. کلاس IPEndPoint از فضای نام System.Net به تعریف این نقطه پایانی کمک می‌کند.



## ایجاد سوکت

در C#، با استفاده از .NET، کلاس System.Net.Sockets موجود در فضای نام System.Net.Sockets سوکت‌ها ابزار اصلی برای ایجاد و مدیریت سوکت‌ها است. یک نمونه سوکت جدید را می‌توان با ارائه سه اطلاعات کلیدی ایجاد کرد: خانواده آدرس، نوع سوکت و نوع پروتکل.

# پذیرش اتصالات

پس از تشخیص یک اتصال ورودی، سرور می‌تواند آن را با استفاده از متدهای `Accept` پذیرد. این متدهای مسدودکننده است؛ منتظر می‌ماند تا یک کلاینت متصل شود.

## ماهیت مسدودکننده `Accept`

متدهای `Accept`، هنگامی که روی یک سوکت سرور فراخوانی می‌شود، رشته فعلی اجرا را مسدود می‌کند تا زمانی که یک کلاینت سعی در اتصال کند. هنگامی که یک درخواست اتصال می‌رسد، یک سوکت جدید اختصاص داده شده به کلاینت متصل شونده را برمی‌گرداند.

```
Socket clientSocket = serverSocket.Accept();
```

این سوکت جدید `clientSocket` در مثال (به عنوان کانال ارتباطی بین سرور و کلاینت خاص عمل می‌کند.

## مدیریت چندین اتصال با استفاده از ترد

در یک سناریوی دنیای واقعی، یک سرور معمولاً به چندین کلاینت به طور همزمان خدمات می‌دهد. یک رویکرد برای دستیابی به این امر استفاده از ترد است. با هر اتصال جدید، یک ترد جدید می‌تواند برای مدیریت درخواست‌های کلاینت ایجاد شود، که به ترد اصلی سرور اجازه می‌دهد به گوش دادن برای سایر اتصالات ورودی ادامه دهد.

# تبادل داده با کلاینت‌ها

جوهر ارتباط سرور-کلاینت، تبادل داده است. پس از برقراری اتصال بین یک سرور و یک کلاینت، یک کانال ارتباطی دوطرفه تشکیل می‌شود که اجازه می‌دهد داده‌ها در هر دو جهت جریان یابد. این داده‌ها می‌توانند هر چیزی را نشان دهند، از پیام‌های متنی ساده گرفته تا داده‌های باینری پیچیده، مانند فایل‌ها یا اشیاء سریالی شده.

## دریافت داده از کلاینت‌ها

داده‌ها از کلاینت می‌توانند با استفاده از متدهای Receive دریافت شوند. مهم است که یک بافر برای نگهداری داده‌های ورودی آماده کنید:

```
byte[] buffer = new byte[1024]; // بافر 1 کیلوبايتی  
int bytesRead = clientSocket.Receive(buffer);  
  
string receivedMessage = Encoding.UTF8.GetString(buffer, 0,  
bytesRead);  
  
Console.WriteLine($"Received: {receivedMessage}");
```

در این کد، متدهای Receive تا زمانی که داده‌ای از کلاینت دریافت شود، مسدود می‌شود. مقدار برگشتی نشان‌دهنده تعداد بایت‌های خوانده شده است. سپس این بایت‌ها را دوباره به یک رشته تبدیل می‌کنیم تا آن را پردازش یا نمایش دهیم.

## ارسال داده به کلاینت‌ها

پس از اینکه یک سرور اتصال کلاینت را پذیرفت، می‌تواند با استفاده از متدهای Send روی سوکت اختصاصی کلاینت، داده‌ها را به کلاینت ارسال کند:

```
byte[] messageBytes = Encoding.UTF8.GetBytes("Welcome to the  
server!");  
  
clientSocket.Send(messageBytes);
```

در اینجا، داده (یک پیام رشته‌ای (ابتدا با استفاده از کدگذاری UTF-8) به یک آرایه بایت تبدیل می‌شود، و سپس با استفاده از متدهای Send به کلاینت ارسال می‌شود.

# مدیریت جلسات کلاینت

مدیریت جلسات کلاینت یک جزء حیاتی از برنامه‌نویسی سوکت سمت سرور است. یک جلسه نشان‌دهنده مدت زمان تعامل بین سرور و یک کلاینت است. مدیریت مؤثر جلسه امکان ردیابی، نگهداری و عملیات روی داده‌های خاص کلاینت پایدار را فراهم می‌کند، تضمین تجربه کاربری بدون وقفه، افزایش امنیت و بهینه‌سازی منابع سرور.

2

## ذخیره داده‌های جلسه

یک دیکشنری همزمان برای ذخیره داده‌های مرتبط با جلسه ایده‌آل است زیرا عملیات امن از نظر ترد را ارائه می‌دهد.

```
ConcurrentDictionary sessionData = new ConcurrentDictionary();
```

برای هر کلاینت، می‌توانید داده‌های خاص جلسه را ذخیره و بازیابی کنید:

```
sessionData.TryAdd(sessionId, new ClientSessionData());
```

```
ClientSessionData data =
(ClientSessionData)sessionData[sessionId];
```

1

## شناسایی جلسات کلاینت

هر اتصال کلاینت نیاز به یک شناسه منحصر به فرد دارد. این می‌تواند ترکیبی از آدرس IP او پورت کلاینت، یا یک شناسه جلسه تولید شده سفارشی باشد.

```
Socket clientSocket = serverSocket.Accept();
string clientId = $"{clientSocket.RemoteEndPoint}";
Console.WriteLine($"New session initiated: {clientId}");
```

# خلاصه

با پایان سفر مقدماتی ما به برنامه‌نویسی سوکت با C# 12 و .NET 8، مشخص است که دنیای برنامه‌های شبکه‌ای گستردگی و پویا است. ما سنگ‌های بنیادی را گذاشته‌ایم، پیچیدگی‌های ارتباط کلاینت-서ور را کاوش کرده‌ایم، به چالش‌های مدیریت چندین کلاینت پرداخته‌ایم و مدیریت خطای قوی را تضمین کرده‌ایم.

در حالی که این فصل یک معرفی جامع ارائه داد، چشم‌انداز برنامه‌نویسی سوکت و برنامه‌های شبکه‌ای همچنان در حال تکامل است. با ابزارها و تکنیک‌های معرفی شده در اینجا، شما برای کاوش عمیق‌تر در زمینه‌های تخصصی‌تر شبکه یا گسترش به انتزاع‌های سطح بالاتر ارائه شده توسط C# و .NET آماده هستید.

دانش برنامه‌نویسی سوکت در C# که در این فصل ایجاد شد، پایه مهمی برای بحث‌های آینده می‌گذارد. این را با مهارت‌های اساسی برای کاوش در مفاهیم پیشرفته شبکه، ارتباط داده کارآمد و توسعه برنامه‌های مقیاس‌پذیر مجهز می‌کند.

# برنامه‌نویسی ناهمگام با Async/Await

به فصل مهمی از سفر شما در برنامه‌نویسی شبکه با استفاده از C# خوش آمدید، جایی که به برنامه‌نویسی ناهمگام با استفاده از کلیدواژه‌های `async` و `await` پردازیم.

```
#async function fetchData({:  
  -async function ()  
    const funtion {.  
      const result = await ...  
      await result = await ...);  
}
```



```
#async function<ption processData {  
  -awain {  
    cosit function ();  
    cost processElt();  
    eost await =...);  
}
```



```
#async function fitems {.  
  -swal function {  
    atwait updatiUI ({  
      fost updateIUI();  
      awair =...);  
}
```

# مقدمه

همانطور که در پیچیدگی‌های برنامه‌نویسی شبکه پیش رفته‌اید، یاد گرفته‌اید که چگونه اتصالات قوی ایجاد کنید، داده‌ها را منتقل کنید و پروتکل‌های مختلف شبکه را مدیریت کنید. اکنون به نقطه‌ای رسیده‌ایم که کارایی و پاسخگویی اهمیت بالایی پیدا می‌کند.

در این فصل، قدرت و ظرافت الگوهای برنامه‌نویسی ناهمگام C# را بررسی خواهیم کرد که عملکرد را بهبود می‌بخشد و پاسخگویی برنامه‌ها را حتی در مواجهه با عملیات‌های شبکه‌ای پیچیده حفظ می‌کند.



# ماهیت برنامه‌های شبکه

## عملیات زمان‌بر

برنامه‌های شبکه با عملیات ذاتی زمان‌بر و غیرقابل پیش‌بینی سروکار دارند. داده‌ها ممکن است در سراسر قاره‌ها سفر کنند و زمان ارسال درخواست و دریافت پاسخ می‌تواند قابل توجه باشد.

## انتظار برای داده‌ها

برنامه شما ممکن است چرخه‌های ارزشمند CPU را صرف انتظار برای جابجایی داده‌ها در شبکه کند. اینجاست که برنامه‌نویسی ناهمگام می‌درخشد.

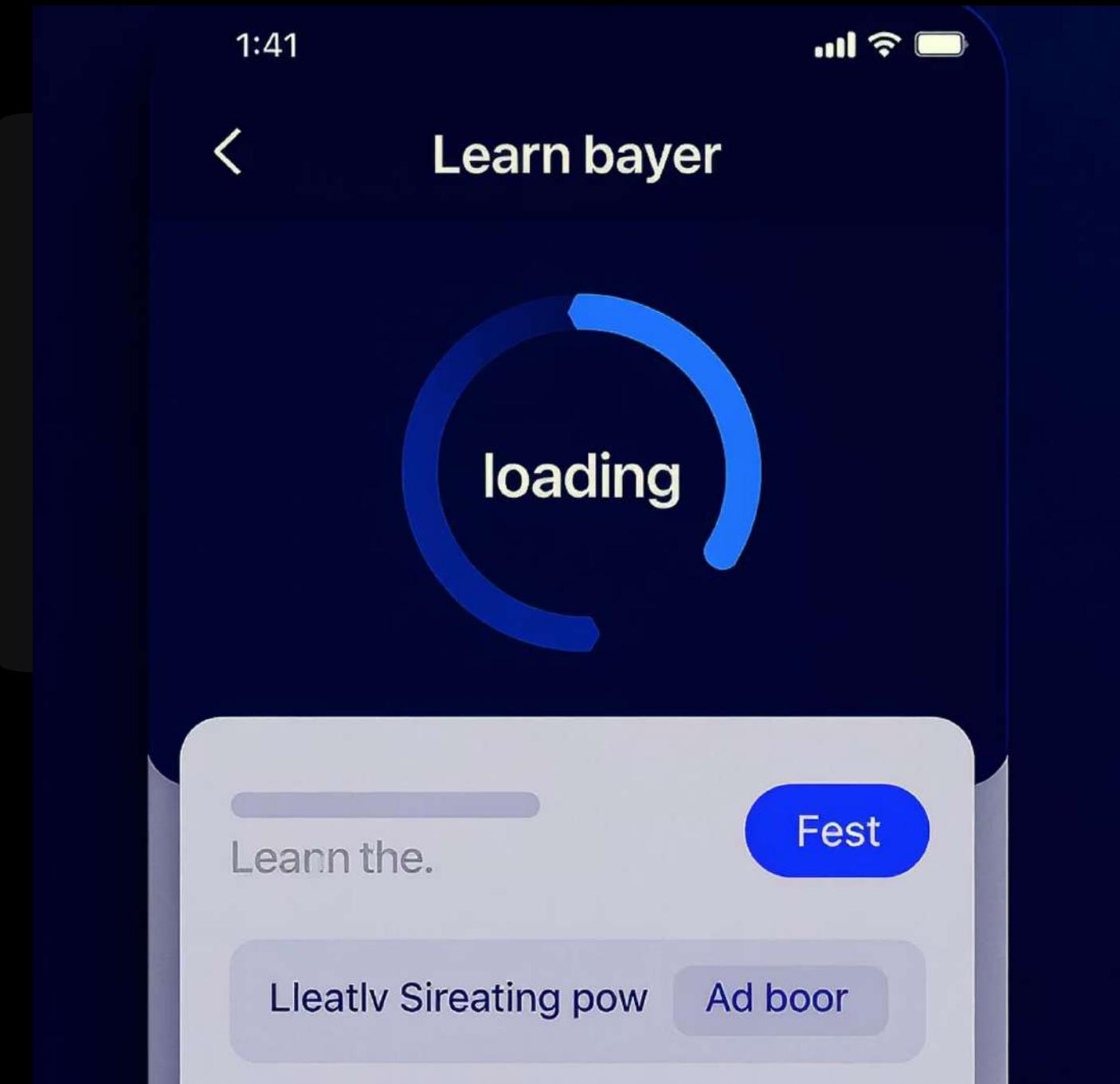
## کلیدواژه‌های await و gasync

با کلیدواژه‌های await و gasync که در C 5.0 معرفی شدند، ما مجهر به نوشتن کدهایی هستیم که هم کارآمد و هم به راحتی قابل خواندن هستند، شبیه به سادگی کد همگام اما با اجرای غیرمسود کننده.

# سناریوی کاربردی

تصور کنید سناریویی که برنامه شما باید مقادیر زیادی داده را از یک سرور راه دور دریافت کند یا منتظر دانلود یک فایل از طریق یک اتصال کند باشد.

مسدود کردن رابط کاربری یا مصرف غیرضروری منابع نخ در حین تکمیل این عملیات‌ها منجر به تجربه کاربری ضعیف و استفاده ناکارآمد از منابع می‌شود.



<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

# اهداف این فصل

1

درک `await` و `gasync`

درک خواهید کرد که چگونه از `gasync` و `await` برای انجام عملیات‌های شبکه بدون پیچیدگی‌های سنتی مرتبط با برنامه‌نویسی ناهمگام استفاده کنید.

2

کد کارآمدتر و ساده‌تر

قادر خواهید بود کدی بنویسید که نه تنها کارآمدتر است بلکه ساده‌تر و قابل نگهداری‌تر است.

3

مدیریت استثناهای طولانی

یاد خواهید گرفت که چگونه استثناهای را در کد ناهمگام مدیریت کنید، پیشرفت را گزارش دهید و عملیات‌های شبکه‌ای طولانی‌مدت را به طور مناسب لغو کنید.

# موضوعات اصلی

## درک Async/Await عملیات‌های ناهمگام

بررسی عمیق نحوه کار کلیدوازه‌های  
چگونگی و await و gasync  
پیاده‌سازی عملیات‌های ناهمگام

## معرفی برنامه‌نویسی ناهمگام

آشنایی با مفاهیم اساسی و اهمیت  
برنامه‌نویسی ناهمگام در برنامه‌های  
شبکه

## استراتژی‌های نوشتن کد ناهمگام

تکنیک‌ها و بهترین شیوه‌های نوشتن کد ناهمگام کارآمد و قابل نگهداری

<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>



# معرفی برنامه‌نویسی ناهمگام

در اجرای همگام سنتی، یک نخ مسدود می‌شود یا منتظر می‌ماند تا عملیات تکمیل شود قبل از اینکه به وظیفه بعدی برود، که منجر به استفاده ناکارآمد از منابع و تجربه کاربری کند می‌شود. برنامه‌نویسی ناهمگام، از طرف دیگر، به نخ اجرا امکان می‌دهد وظایف دیگر را در حین انتظار برای تکمیل عملیات شبکه انجام دهد، بنابراین استفاده بهتر از منابع سیستم و بهبود پاسخگویی برنامه را فراهم می‌کند.

# پیادهسازی برنامه‌نویسی ناهمگام در C#

## استفاده از await

کلیدواژه await کمپرس برای فراخوانی این متدهای ناهمگام استفاده می‌شود، که به متدهای اجرا می‌دهد اجرای خود را متوقف کند تا زمانی که وظیفه منتظر تکمیل شود بدون مسدود کردن نخ.

## نشانه‌گذاری متدها با async

وقتی یک متدهای ناهمگام می‌شود، حاوی عملیات‌های ناهمگام است و یک Task<T> یا Task را برمی‌گرداند.

## کلیدواژه‌های await و gasync

در C#، برنامه‌نویسی ناهمگام عمدهاً با استفاده از کلیدواژه‌های await و gasync انجام می‌شود، که به طور ظرفی در زبان و محیط اجرایی یکپارچه شده‌اند.

در مقایسه با الگوهای برنامه‌نویسی ناهمگام قدیمی‌تر، این مدل مدیریت خطای انتشار استثنای مدیریت زمینه همگام‌سازی را ساده می‌کند. در نتیجه توسعه‌دهندگان می‌توانند کد خواناتر و قابل نگهداری‌تری بنویسند، که برای وظایف پیچیده برنامه‌نویسی شبکه در محیط‌های .NET ضروری است.

A chapter titled "Introduction to C# Concurrency and Synchronization" is available online at [www.conyandcony.com](http://www.conyandcony.com). It includes a detailed explanation of the various synchronization mechanisms available in C#.

In addition, there is a chapter titled "Parallel Programming with C# 5.0" which provides an introduction to parallel programming and the Task Parallel Library (TPL).

1. Create  
multiple tasks  
using `Task.Factory.StartNew`  
or `Task.Run`.

2. Wait  
until all  
tasks have  
completed  
using `Task.WaitAll`.

3. Get  
the results  
of each task  
using `Task.Result`.

4. If  
any task  
fails, catch  
the exception  
using `try`/`catch`.

5. Dispose  
of the tasks  
using `Task.Dispose`.

6. Finally,  
return the  
results of  
all tasks.

## روش‌های اولیه

در ابتدا، C# و .NET پشتیبانی اساسی برای عملیات‌های ناهمگام از طریق مکانیسم‌هایی مانند الگوی `AsyncResult` و متدهای `BeginInvoke` و `EndInvoke` ارائه می‌دادند. این رویکردهای اولیه عملی بودند اما اغلب منجر به کد پیچیده و سخت‌خوان می‌شدند، به ویژه هنگام برخورد با عملیات‌های ناهمگام تودرتو یا چندگانه.

## تحول با #C 5.0

با انتشار #C 5.0 و .NET Framework 4.5، چشم‌انداز برنامه‌نویسی ناهمگام تحول اساسی یافت با معرفی کلیدوازه‌های `await` و `async` و `await` مدل جدید به طور قابل توجهی نوشتمن و در کد ناهمگام را ساده کرد، به توسعه‌دهندگان اجازه داد عملیات‌های ناهمگام را به شیوه‌ای بنویسند که بسیار شبیه به کد همگام است، بنابراین پیچیدگی را کاهش داده و خوانایی را بهبود می‌بخشد.

# نقش برنامه‌نویسی ناهمگام در برنامه‌های شبکه

## پاسخگویی برنامه

با پیاده‌سازی برنامه‌نویسی ناهمگام، توسعه‌دهندگان می‌توانند اطمینان حاصل کنند که یک برنامه حتی هنگام برخورد با اتصالات شبکه کند یا انتقال داده‌های بزرگ، پاسخگو و کارآمد باقی می‌ماند.

## Traffیک شبکه بالا

نقش برنامه‌نویسی ناهمگام در برنامه‌های شبکه به ویژه در سناریوهایی که شامل سطوح بالای ترافیک شبکه و پردازش داده‌ها هستند، مشهود است. به جای توقف اجرا تا زمان دریافت پاسخ شبکه، یک رویکرد ناهمگام به برنامه اجازه می‌دهد به پردازش وظایف دیگر ادامه دهد.



## کارایی در عملیات‌های I/O

برنامه‌نویسی ناهمگام نقش حیاتی در توسعه و عملکرد برنامه‌های شبکه دارد. در شبکه، جایی که برنامه‌ها اغلب منتظر ارسال یا دریافت داده‌ها هستند، کارایی مدیریت این عملیات‌های I/O می‌تواند تأثیر قابل توجهی بر عملکرد کلی و تجربه کاربر داشته باشد.

## مقیاس‌پذیری

برنامه‌نویسی ناهمگام امکان استفاده بهتر از منابع و مقیاس‌پذیری در برنامه‌های شبکه را فراهم می‌کند. با آزاد کردن نخ‌هایی که در غیر این صورت در طول عملیات‌های I/O مسدودکننده بیکار می‌مانند، این نخ‌ها می‌توانند برای اهداف دیگر استفاده شوند.

# چالش‌های برنامه‌نویسی ناهمگام

برنامه‌نویسی ناهمگام تحول بزرگی در توسعه برنامه‌های پاسخگو بوده است، اما علی‌رغم مزایای آن، چندین چالش را معرفی می‌کند که توسعه و اشکال‌زدایی را پیچیده می‌کند.

- مدیریت جریان‌های کنترل پیچیده
- شرایط مسابقه و بن‌بست‌ها
- استثناهای پرتاب شده در وظایف ناهمگام
- اشکال‌زدایی کد ناهمگام
- بهینه‌سازی عملکرد برنامه‌های ناهمگام



<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

# دامهای رایج

2

## مدیریت استثنای در کد ناهمگام

اگر به درستی منتظر نماند یا مدیریت نشود، استثناهای پرتاب شده در وظایف ناهمگام همیشه در بلوک‌های try-catch گرفته نمی‌شوند، منجر به مشاهده نشده که می‌توانند باعث رفتار غیرمنتظره یا خرابی برنامه شوند.

1

## سوء استفاده از await و gasync

برخی توسعه‌دهندگان ممکن است کلیدواژه `gasync` را به هر متدهای اعمال کنند، منجر به سربار غیرضروری یا سوءاستفاده از کلیدواژه `await`، که منجر به بنبست‌ها یا استفاده ناکارآمد از منابع می‌شود.

4

## حفظ وضوح و خوانایی کد

توسعه‌دهندگان ممکن است در حفظ وضوح و خوانایی کد هنگام استفاده از برنامه‌نویسی ناهمگام مشکل داشته باشند، به ویژه هنگام برخورد با فراخوانی‌های ناهمگام تودرتو یا جریان کنترل پیچیده.

3

## مدیریت منابع

عملیات‌های ناهمگام می‌توانند منجر به عملیات‌های همزمان بیشتری شوند، افزایش بار روی منابع سیستم مانند اتصالات شبکه یا حافظه. اگر به دقت مدیریت نشود، این می‌تواند منجر به نشت منابع یا رقابت منابع شود.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# درک زمینه همگامسازی

در پروژه‌های شبکه C#، درک زمینه همگامسازی برای مدیریت مؤثر همزمانی عملیات‌های ناهمگام بسیار مهم است. زمینه همگامسازی در مسابقه، بنبست‌ها یا بهروزرسانی‌ها از دهد صفت موارد کاری، پیام‌ها یا مدیریت‌کننده‌های رویداد Windows.NET اجازه می‌دهد به زمینه یا نخ اصلی برگردند، مانند نخ الادر یک برنامه WPF یا Forms

این به ویژه در برنامه‌های شبکه مهم است که به روزرسانی‌های الایا دسترسی به منابع باید با پاسخ‌های شبکه همگام شوند تا از شرایط مسابقه، بنبست‌ها یا بهروزرسانی‌ها از یک نخ غیر الایا، که می‌تواند باعث استثنای شود، جلوگیری شود.

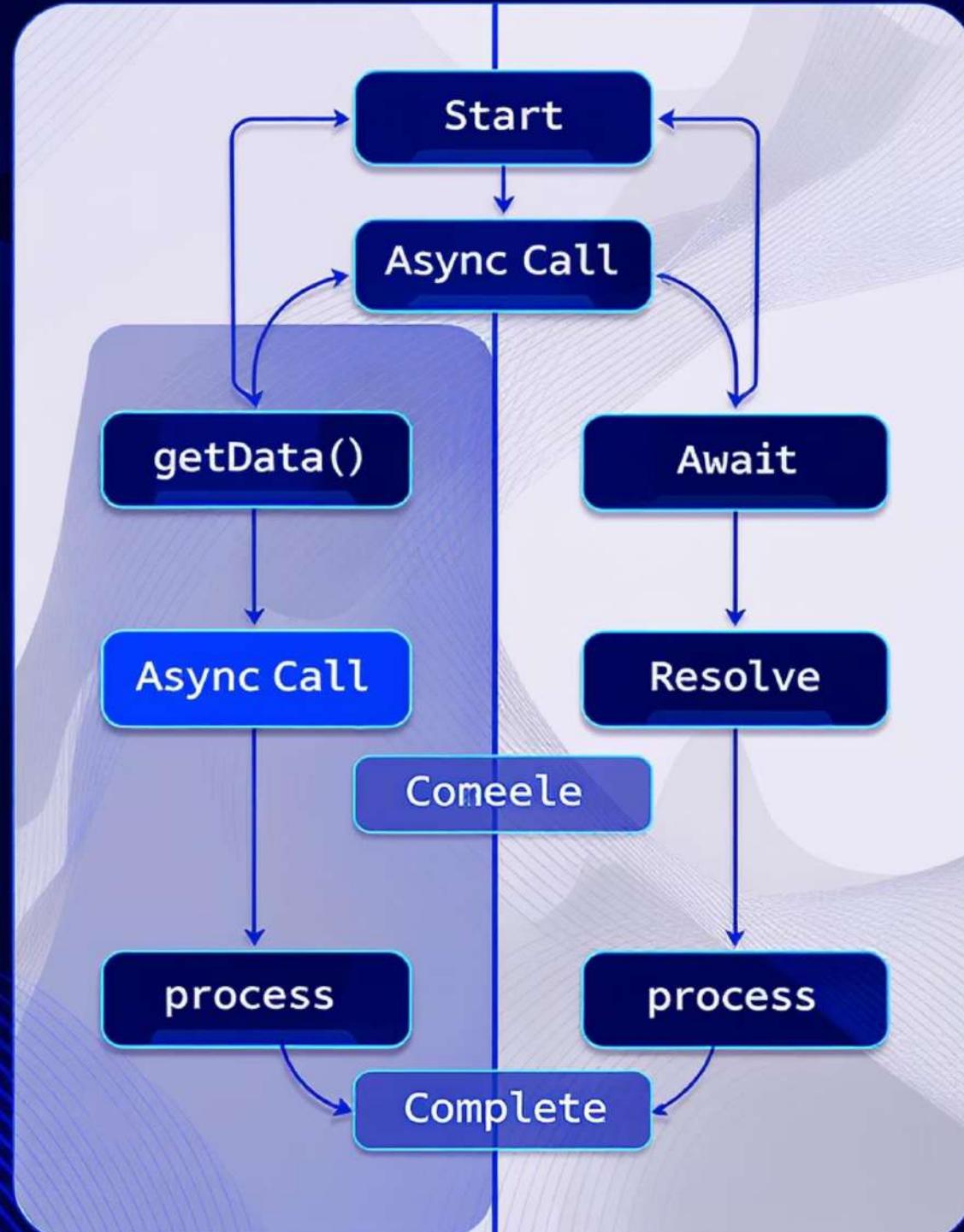
توسعه‌دهندگان باید درک کنند که چگونه زمینه همگامسازی توسط کلیدواژه‌های `await` و `async` گرفته و استفاده می‌شود تا اجرای ادامه‌ها (کدی که پس از یک عملیات اجرا می‌شود) را به زمینه اصلی برگرداند.

# درک عملیات‌های ناهمگام `async/await`

درک عملیات‌های ناهمگام و الگوی `async/await` توسعه برنامه‌های مدرن، کارآمد و مقیاس‌پذیر C# و .NET ضروری است. برنامه‌نویسی ناهمگام به ویژه در برنامه‌نویسی شبکه، جایی که عملیات‌هایی مانند درخواست‌های وب، O/Aفایل و تراکنش‌های پایگاه داده می‌توانند تأثیر قابل توجهی بر عملکرد و پاسخگویی داشته باشند، اهمیت فزاینده‌ای پیدا کرده است.

کلیدواژه‌های `await` و `async` در C# برنامه‌نویسی ناهمگام را تسهیل می‌کنند با اجازه دادن به توسعه‌دهندگان برای نوشتن کدی که هم کارآمد و هم خواندن آن آسان است و بسیار شبیه به ساختارهای کد همگام سنتی است.

## Async dit Flow



# اصول اساسی Async/Await

در C#، کلیدواژه‌های `async` و `await` سنگ بنای برنامه‌نویسی ناهمگام را تشکیل می‌دهند، که به توسعه دهندگان امکان می‌دهد که تمیزتر و خواناتری برای عملیات‌های ناهمگام بنویسند.

کلیدواژه `async` یک متدهای ناهمگام تعریف می‌کند، نشان می‌دهد که متدهای حاوی عملیاتی است که ممکن است شامل انتظار باشد، مانند فراخوانی‌های شبکه یا `0/0` افایل، بدون مسدود کردن نخ اجرا.

وقتی با `await` نشانه‌گذاری می‌شود، یک متدهای `Task<T>` یا `ValueTask<T>` نشان‌دهنده کار در حال انجام است. کلیدواژه `await`، که در متدهای `async` استفاده می‌شود، اجرای متدهای `Task` را متوقف می‌کند تا زمانی که `Task` متنظر تکمیل شود، اجازه می‌دهد عملیات‌های دیگر بدون قفل کردن نخ اصلی به طور همزمان اجرا شوند.

# اصلاح‌کننده `async`

اصلاح‌کننده متدهای `async` در C# نشان می‌دهد که یک متدهایی است که برگرداندن یک `Task<T>` یا `ValueTask<T>` را نمایند. این متدهایی عبارت‌های `await` و `TaskAwaiter` را در خود دارند. همچنان که در آن متدهایی که برگردانند یک `Task<T>`، این متدهایی نیز ممکن است هنوز کامل نشده باشند.

حضور `async` نوع برگشتی متدهای `Task<T>` یا `ValueTask<T>` را تغییر می‌دهد، که به آن امکان می‌دهد که نشان‌دهنده کار در حال انجامی است که ممکن است هنوز کامل نشده باشد.

این رویکرد برای توسعه برنامه‌های غیرمسدودکننده ضروری است، به ویژه در برنامه‌های الایا سرویس‌هایی که پاسخگویی و مقیاس‌پذیری حیاتی است.

# کلیدوازه await

کلیدوازه `#`await در C# یک ویژگی محوری برنامه‌نویسی ناهمگام است که در ترکیب با اصلاح‌کننده `async` استفاده می‌شود. این به متدهای فعلی اجازه می‌دهد اجرای خود را متوقف کند تا زمانی که وظیفه ناهمگام منتظر تکمیل شود بدون مسدود کردن نخ فراخواننده.

کنترل در طول این انتظار به فراخواننده بر می‌گردد، که امکان اجرای همزمان عملیات‌های دیگر را فراهم می‌کند. این مکانیسم برای توسعه برنامه‌های پاسخگو ضروری است، به ویژه هنگام برخورد با وظایف مبتنی بر API‌ها، که ممکن است در طول این انتظار اتفاق بپزدید. این مکانیسم برای خواندن فایل‌ها، عملیات‌های پایگاه داده یا انجام درخواست‌های وب.



```
    {
        await keyword;
    }
    'await "art (enpint';
    {
        the 'art awnifile (app);
        { no "rsusi");
        }
    }
}
```

<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

# Strategies for Writing Asynchronous Code

## Using Async/Await

```
async asuc {  
    Slimple: funcion: {;  
        cable fuction(asya):  
    },
```

## Leveraging Promises

```
swrape fromise: {  
    tiple frome/ise();  
    cndlent ased:  
        catica(1el 2f, promises)  
}
```

## Employing Callbacks

```
funadiciin awrat: {  
    tiple frome/ise();  
};
```

# استراتژی‌های نوشتن کد ناهمگام

نوشتن کد ناهمگام برای توسعه نرم‌افزار مدرن ضروری است، به ویژه هنگام ساخت برنامه‌ها و سرویس‌های مقیاس‌پذیر و پاسخگو. مدل برنامه‌نویسی ناهمگام در زبان‌های مانند C# به توسعه‌دهندگان اجازه می‌دهد عملیات‌های غیرمسدودکننده را انجام دهند، مانند درخواست‌های وب، O/Aفایل و تراکنش‌های پایگاه داده، بنابراین پاسخگویی رابط کاربری و مقیاس‌پذیری سرویس‌های بک‌اند را بهبود می‌بخشد.

با این حال، بهره‌برداری مؤثر از این مدل نیازمند استراتژی‌های دقیق برای مدیریت پیچیدگی‌های ذاتی کد ناهمگام، مانند بنبست‌های بالقوه، حفظ وضوح کد و مدیریت استثناهای است.

# چه زمانی از `async/await` استفاده کنیم



## پاسخگویی API

برنامه‌نویسی ناهمگام اگر برنامه شما یک رابط کاربری دارد و نیاز دارد پاسخگویی آن را در حین انجام عملیات‌های شبکه حفظ کنید، بسیار مهم است. این اطمینان می‌دهد که برنامه شما همچنان می‌تواند تعاملات کاربر را مدیریت کند، حتی زمانی که مشغول دریافت داده از وب است.



## فراخوانی‌های شبکه طولانی‌مدت

باشد هر زمان که فراخوانی `await` و `Async` دانلود فایل‌ها یا انجام هر عملیات شبکه‌ای که بیش از یک چشم به هم زدن آنها از یخ زدن برنامه شما در حین انتظار طول می‌کشد را انجام می‌دهید برای پاسخ شبکه جلوگیری می‌کنند.



## سایر عملیات‌های I/O

هنگام خواندن یا نوشتن فایل‌ها در دیسک یا انتظار برای یک کوئری پایگاه داده طولانی‌مدت، تابع یا رویه ذخیره شده، استفاده از `await` و `gasync` می‌تواند عملکرد کمک کند و به کاربر اجازه دهد تجربه بهتری در استفاده از برنامه شما داشته باشد.



## مقیاس‌پذیری

هنگام نوشتن کد سمت سرور، مانند یک سرویس وب، استفاده از `gasync` و `await` می‌تواند مقیاس‌پذیری را بهبود بخشد. به سرور شما اجازه می‌دهد درخواست‌های بیشتری را همزمان مدیریت کند با عدم اشغال نخ‌ها در انتظار برای تکمیل عملیات‌های I/O.

<https://www.linkedin.com/in/mhramini/>

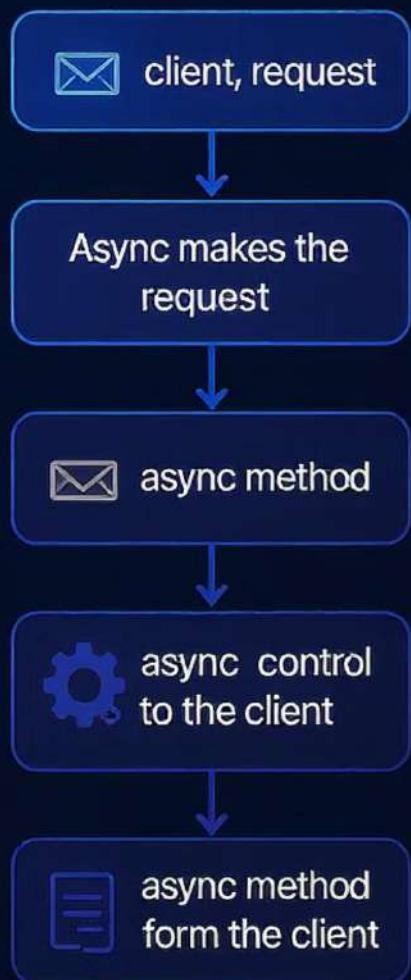
<https://github.com/MohamadHoseinRoohiAmini>

# طراحی متدهای ناهمگام

طراحی متدهای ناهمگام در C# یک ویژگی قدرتمند برای بهبود مقیاس پذیری و پاسخگویی برنامه ها است، به ویژه در سناریوهای شامل عملیات های مبتنی بر IoT، مانند درخواست های وب، دسترسی به فایل و تراکنش های پایگاه داده.

با استفاده از کلیدواژه های `await` و `async` توسعه دهندگان می توانند کد ناهمگامی بنویسند که تقریباً به همان اندازه خواندن و نوشتن کد همگام ساده است. این الگوی طراحی به یک متدهای اجازه می دهد بدون مسدود کردن نخی که روی آن اجرا می شود، به صورت ناهمگام اجرا شود.

## ASYNC METHOD



<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

# ناهمگام در تمام مسیر



## منطق برنامه

این کلیدوازه‌ها در اجرای عملیات‌های مبتنی بر `O`، مانند درخواست‌های HTTP، انتقال فایل یا کوئری‌های پایگاه داده، به صورت ناهمگام برای جلوگیری از مسدود کردن نخ اصلی بسیار مهم هستند.



## عملیات‌های شبکه

اتخاذ استراتژی "ناهمگام در تمام مسیر" به معنای اعمال مداوم اصول برنامه‌نویسی ناهمگام در سراسر کد شما هر زمان که یک عملیات ناهمگام را آغاز می‌کنید است.



## رابط کاربری

در توسعه نرم‌افزار شبکه با C#، استفاده جامع از `await` و `gasync` از رابط کاربری تا پایین‌ترین عملیات‌های شبکه، برای افزایش پاسخگویی و عملکرد برنامه بسیار مهم است.



## دسترسی به داده

این رویکرد به برنامه شما اجازه می‌دهد در حین انتظار برای پاسخ‌های شبکه وظایف دیگر را انجام دهد، از یخ زدن برنامه و گلوگاه‌های سرور جلوگیری می‌کند.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>



# اجتناب از `async void`

یک بهترین شیوه رایج در برنامه‌نویسی ناهمگام C# اجتناب از متدهای `void` است، به جز در سناریوهای خاص مانند مدیریت کننده‌های `async`. دلیل اصلی این راهنمایی رفتار مدیریت استثنای متدهای `async void` است، که می‌تواند منجر به استثناهای مدیریت نشده‌ای شود که برنامه را خراب می‌کنند.

برخلاف متدهای Task `async`، که در آن استثناهای گرفته می‌شوند و می‌توانند توسط فراخواننده مشاهده و مدیریت شوند، استثناهای پرتاب شده در متدهای `void` زمینه همگام‌سازی منتشر می‌شوند و به راحتی قابل گرفتن نیستند. این رفتار اشکال‌زدایی و مدیریت خطأ را به طور قابل توجهی چالش‌برانگیزتر می‌کند.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# مدیریت استثنای در کد ناهمگام

## مدیریت TaskCanceledException

مدیریت TaskCanceledException در برنامه‌نویسی ناهمگام بسیار مهم است، به ویژه هنگام کار با وظایفی که می‌توانند لغو شوند، مانند عملیات‌های طولانی‌مدت یا درخواست‌های شبکه. این استثنای زمانی پرتاب می‌شود که یک وظیفه لغو می‌شود، معمولاً از طریق استفاده از یک CancellationToken.

## استفاده از try-catch

مدیریت صحیح استثنایها در وظایف ناهمگام شامل استفاده از بلوک‌های try-catch در اطراف دستورات await یا گرفتن استراتژیک استثنایها از وظایف برگشتی است. این رویکرد اطمینان می‌دهد که استثنایها مدیریت می‌شوند و برنامه می‌تواند به طور مناسب به خطاهای پاسخ دهد.

## چالش استثنای ناهمگام

مدیریت صحیح استثنایها در برنامه‌نویسی ناهمگام برای حفظ ثبات برنامه و ارائه یک تجربه کاربری قوی بسیار مهم است. وقتی استثنایها در متدهای ناهمگام به درستی مدیریت نمی‌شوند، می‌توانند منجر به استثنای مدیریت نشده‌ای شود که ممکن است برنامه را خراب کنند یا باعث رفتار نامنظم شوند.

# استفاده کارآمد از منابع

استفاده کارآمد از منابع در برنامه‌نویسی ناهمگام برای ایجاد برنامه‌های مقیاس‌پذیر و با عملکرد بالا حیاتی است. عملیات‌های ناهمگام، به ویژه آنهای که شامل فعالیت‌های ۰/امانند دسترسی به فایل، ارتباطات شبکه یا تراکنش‌های پایگاه داده هستند، باید با دقت مدیریت شوند تا از مصرف غیرضروری منابع جلوگیری شود.

مدیریت کارآمد منابع در وظایف ناهمگام اطمینان می‌دهد که برنامه توان عملیاتی را به حداقل می‌رساند و تأخیر را به حداقل می‌رساند، که یک تجربه کاربری روان حتی تحت بار سنگین ارائه می‌دهد. این شامل کسب استراتژیک منابع درست قبل از نیاز به آنها و آزاد کردن فوری آنها پس از استفاده است، بنابراین احتمال رقابت و اتمام منابع را کاهش می‌دهد.

# خلاصه

## بهترین شیوه‌ها

پیروی از بهترین شیوه‌ها مانند اجتناب از `void`, مدیریت مناسب استثناهای `async void` استفاده کارآمد از منابع برای نوشتن کد ناهمگام قوی و قابل اعتماد ضروری است.

## مزایای `await` و `gasync`

کلیدواژه‌های `await` و `gasync` برنامه‌نویسی ناهمگام را ساده می‌کنند، به توسعه دهنده‌گان اجازه می‌دهند کدی بنویسند که خوانا و نگهداری آن آسان است، در حالی که عملکرد و پاسخگویی را بهبود می‌بخشد.

## اهمیت برنامه‌نویسی ناهمگام

برنامه‌نویسی ناهمگام در C# برای توسعه برنامه‌های پاسخگو و مقیاس‌پذیر ضروری است، به ویژه در برنامه‌نویسی شبکه که با عملیات‌های زمان‌بر سروکار دارد.

با تسلط بر برنامه‌نویسی ناهمگام در C#, توسعه دهنده‌گان می‌توانند برنامه‌هایی بسازند که نه تنها سریع و پاسخگو هستند بلکه قوی و قابل اعتماد نیز هستند، قادر به مدیریت آسان عملیات‌های پیچیده و بارهای بالا.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# چندنخی در برنامه‌های شبکه

این فصل به بررسی چگونگی استفاده از چندنخی برای بهبود کارایی و کارآمدی برنامه‌های شبکه می‌پردازد. با استفاده از `C12#`, خواهیم آموخت چگونه چندین وظیفه را به صورت موازی اجرا کنیم.



<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# مقدمه به چندنخی در برنامه‌های شبکه

## موازی‌سازی

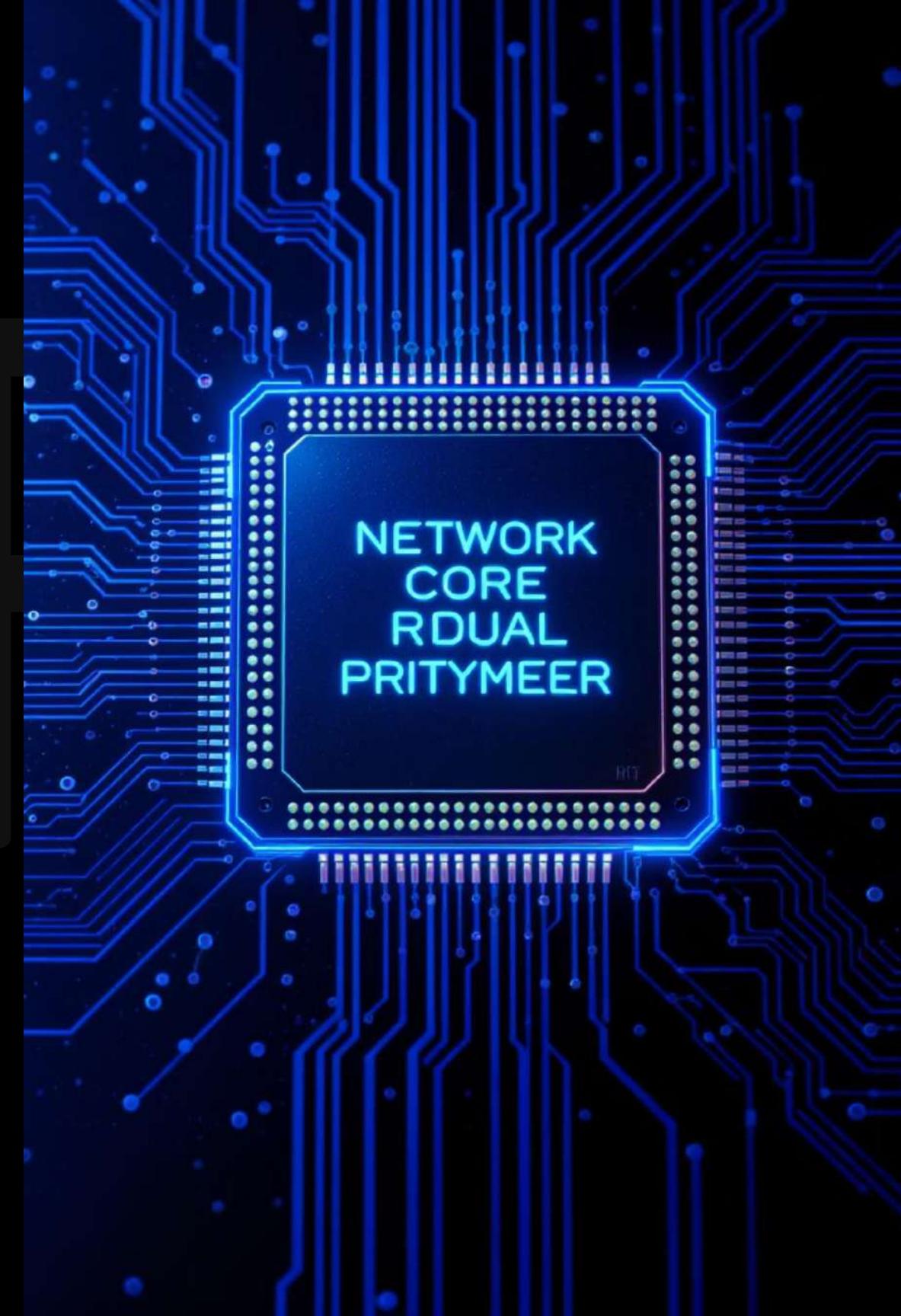
موازی‌سازی به تکنیک تقسیم یک مسئله به وظایفی که می‌توانند به طور همزمان حل شوند و سپس ترکیب نتایج برای دستیابی به نتیجه نهایی اشاره دارد.

## چندنخی چیست؟

چندنخی را می‌توان به عنوان چندین کارگر(نخ)ها (در یک محیط عملیاتی) برنامه شما (تصور کرد که وظایف مختلف را به طور همزمان اجرا می‌کنند).

## مزایای چندنخی

بهبود توان عملیاتی، پاسخگویی بهتر و استفاده بهینه از منابع سیستم از جمله مزایای استفاده از چندنخی در برنامه‌های شبکه است.



<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# موضوعات اصلی این فصل



## مدیریت اتصالات همزمان شبکه با چندنخی

روش‌های مدیریت چندین اتصال همزمان با استفاده از نخ‌ها



## مطالعه موردي: ساخت یک سرور چندنخی

پیاده‌سازی عملی مفاهیم در قالب یک سرور چندنخی کامل



## معرفی چندنخی در برنامه‌های شبکه

آشنایی با مفاهیم اساسی و کاربردهای چندنخی در برنامه‌های شبکه



## پردازش موازی و بهینه‌سازی عملکرد

تکنیک‌های پردازش موازی برای بهبود کارایی برنامه‌های شبکه

## مقدمه به چند نخی در برنامه های شبکه

با افزایش تقاضا برای نرم افزارهای مدرن، توانایی مدیریت چندین عملیات به صورت همزمان برای ساخت برنامه های شبکه کارآمد و مقیاس پذیر ضروری است. چندنخی به یک برنامه شبکه اجازه می دهد تا درخواست های متعدد کاربران را به طور همزمان مدیریت کند و باعث بهبود توان عملیاتی و پاسخگویی می شود.

در این فصل، به معماری برنامه‌های شبکه چندنخی می‌پردازیم و نشان می‌دهیم چگونه C# با پشتیبانی غنی کتابخانه‌ای خود، ایجاد و مدیریت نخ‌ها را تسهیل می‌کند.



# تعریف چندنخی در زمینه شبکه

چندنخی در زمینه برنامه‌نویسی شبکه، به توانایی یک برنامه برای اجرای چندین نخ به صورت همزمان در یک فرآیند واحد اشاره دارد. این امر به ویژه در برنامه‌های شبکه که نیاز به مدیریت کارآمد چندین درخواست همزمان مشتری دارند، بسیار مهم است.

## پشتیبانی #C

خود، پشتیبانی .NET از طریق چارچوب C# قوی برای چندنخی ارائه می‌دهد و ابزارهای مختلف همگام‌سازی مانند قفل‌ها، میوتکس‌ها و سمافورها را برای کمک به مدیریت دسترسی به منابع مشترک بین نخ‌ها فراهم می‌کند.

## بهینه‌سازی منابع CPU

در برنامه‌نویسی شبکه، چندنخی استفاده از منابع CPU را بهینه می‌کند و اطمینان می‌دهد که سرور می‌تواند چندین عملیات را همزمان انجام دهد بدون اینکه منتظر تکمیل یک وظیفه قبل از شروع وظیفه دیگر باشد.

## مسیرهای اجرایی مستقل

هر نخ به عنوان یک مسیر اجرایی جداگانه عمل می‌کند، به برنامه اجازه می‌دهد وظایف متعددی را همزمان انجام دهد، مانند گوش دادن به اتصالات ورودی، پردازش داده‌های مشتری و حفظ اتصالات فعال.

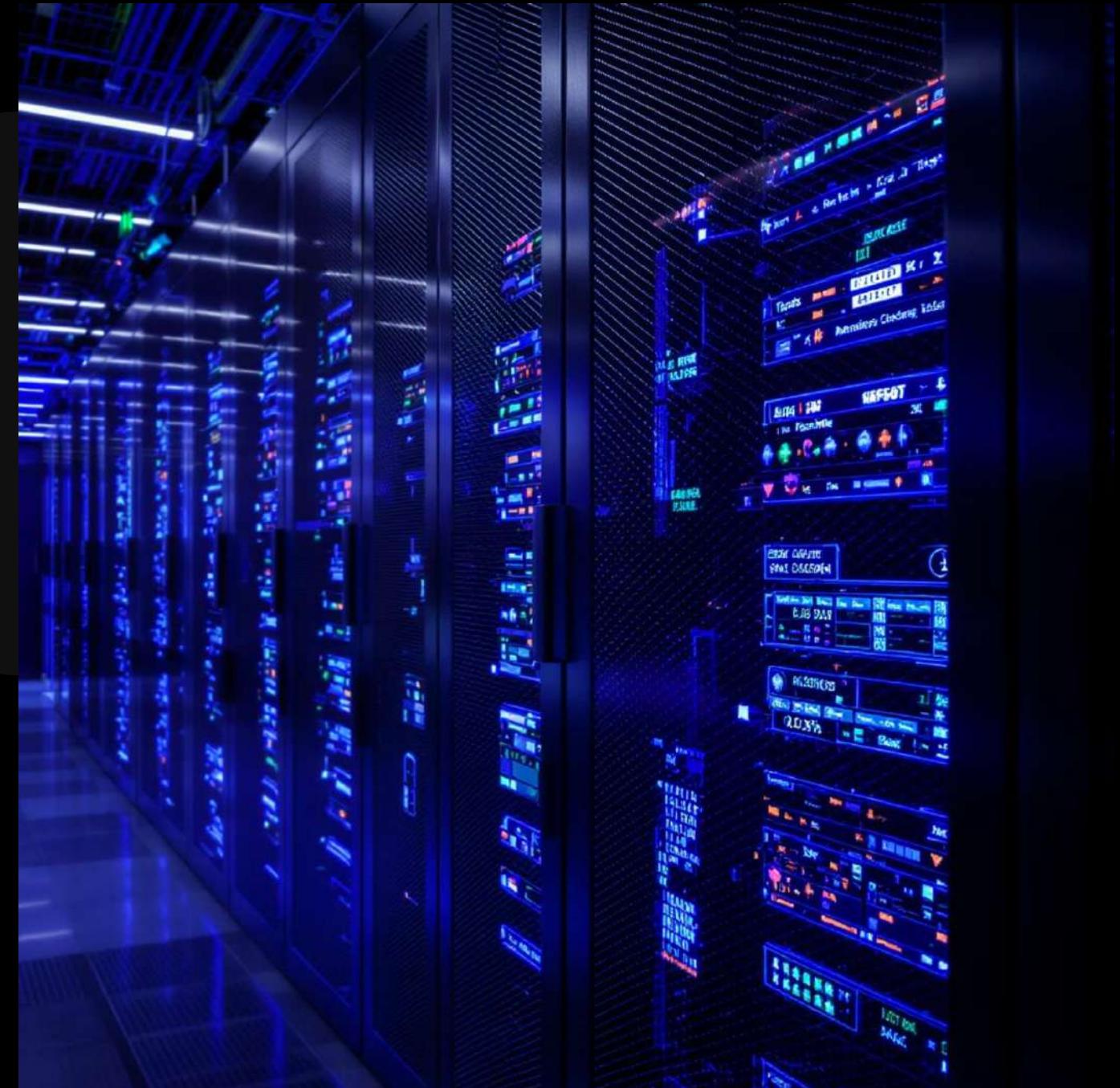
<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# نیاز به چندنخی در برنامه‌های شبکه مدرن

ضرورت چندنخی در برنامه‌های شبکه مدرن از تقاضا برای کارایی و پاسخگویی بیشتر در مدیریت درخواست‌های متعدد مشتری ناشی می‌شود. با داده‌محور و متصل‌تر شدن برنامه‌های شبکه، توانایی پردازش چندین وظیفه به طور همزمان ضروری شده است.

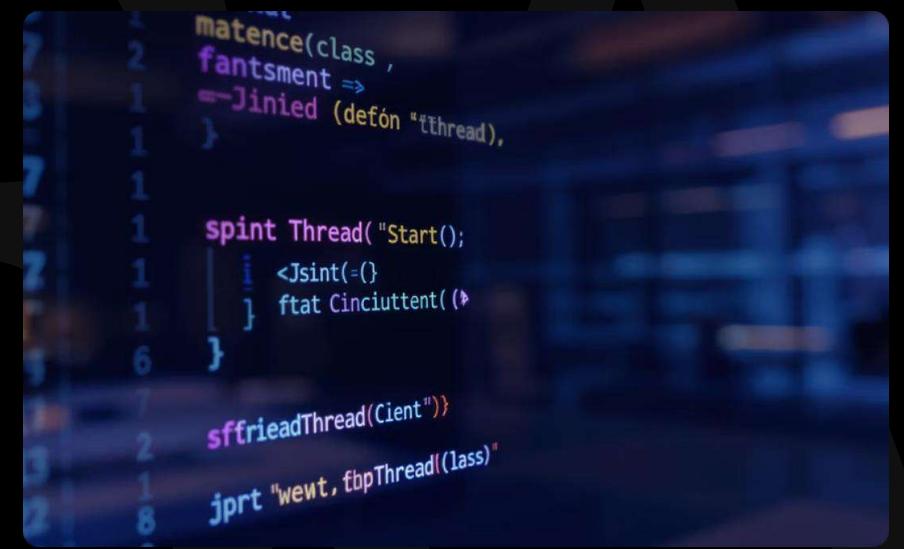
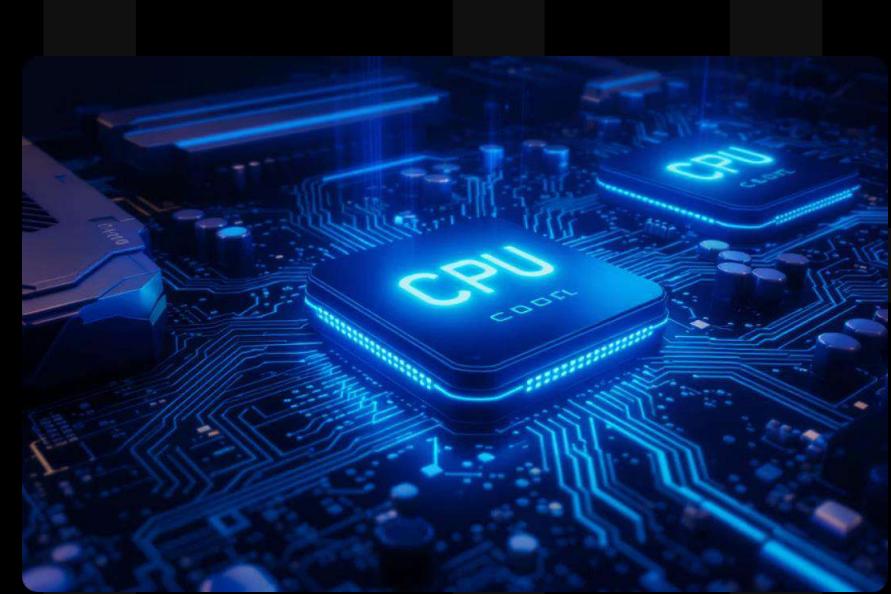
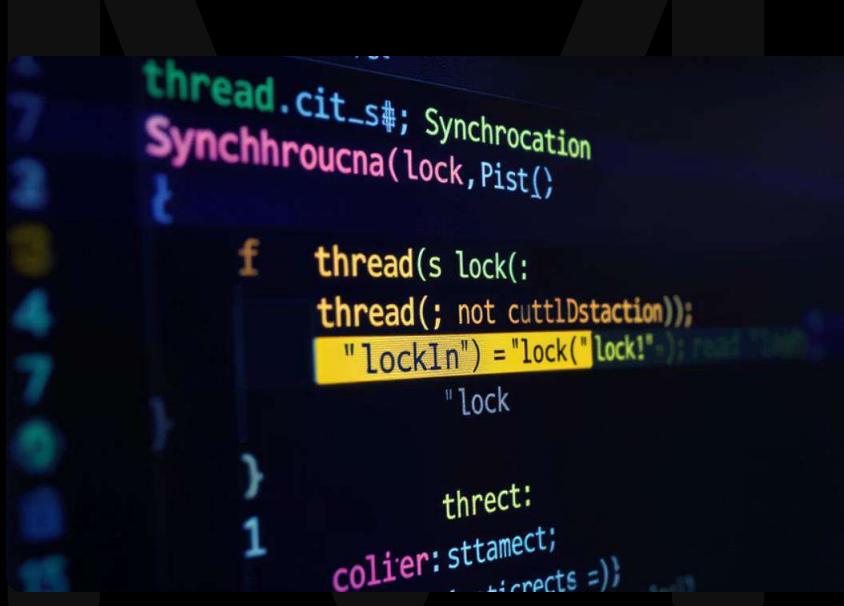
چندنخی به یک سرور اجازه می‌دهد عملیات مختلف را به صورت موازی مدیریت کند، از پردازش داده‌های مشتری گرفته تا مدیریت اتصالات پایگاه داده، بهینه‌سازی استفاده از منابع و کاهش زمان پاسخ.



<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

# مفاهیم اساسی چندنخی

درک مفاهیم اساسی چندنخی برای هر توسعه‌دهنده‌ای که با برنامه‌های شبکه در C# کار می‌کند، ضروری است. در هسته آن، چندنخی شامل ایجاد، اجرا و مدیریت چندین نخ در یک فرآیند برنامه واحد است.



## همگام‌سازی

همگام‌سازی اجزا مانند دستور `#pragma` را برای مدیریت دسترسی به منابع مشترک فراهم می‌کند. این روش برای جلوگیری از فساد داده می‌کند. هنگامی که چندین نخ سعی در دسترسی همزمان به منابع مشترک دارند، ضروری است

موازی‌سازی اجازه می‌دهد چندین نخ به صورت موازی اجرا شوند و استفاده از منابع CPU را بهینه کنند، به خصوص در پردازنده‌های چند هسته‌ای. با این حال، می‌تواند چالش‌هایی مانند شرایط مسابقه و بن‌بست ایجاد کند.

## موازی‌سازی

کلاس `System.Threading.Thread` در C# روشی ساده برای ایجاد نخ‌ها فراهم می‌کند. هر نخ می‌تواند وظایف را به طور مستقل انجام دهد در حالی که حافظه و منابع برنامه را به اشتراک می‌گذارد.

## ایجاد نخ در C#

# مثال کد: ایجاد و شروع یک نخ

```
using System;using System.Threading;public class Example{    public static void Main()    {        Thread newThread = new Thread(DoWork);  
        newThread.Start();    }    static void DoWork()    {        Console.WriteLine("کار در حال شروع است");        Thread.Sleep(5000);        Console.WriteLine("کار به پایان رسید");    }}شبيه‌سازی کار با //
```

در این مثال، متدهای `DoWork` و `Main` را شبیه‌سازی می‌کند. متدهای `newThread` و `Start` را اجرا می‌کند. وقتی `DoWork` فراخوانی می‌شود، نخ اجرای خود را جدا از جریان اصلی برنامه آغاز می‌کند، به نخ اصلی اجازه می‌دهد وظایف خود را ادامه دهد یا نخهای دیگر را مدیریت کند.

<https://www.linkedin.com/in/mohamadroohi/>

<https://github.com/MohamadHoseinRoohiAmini>

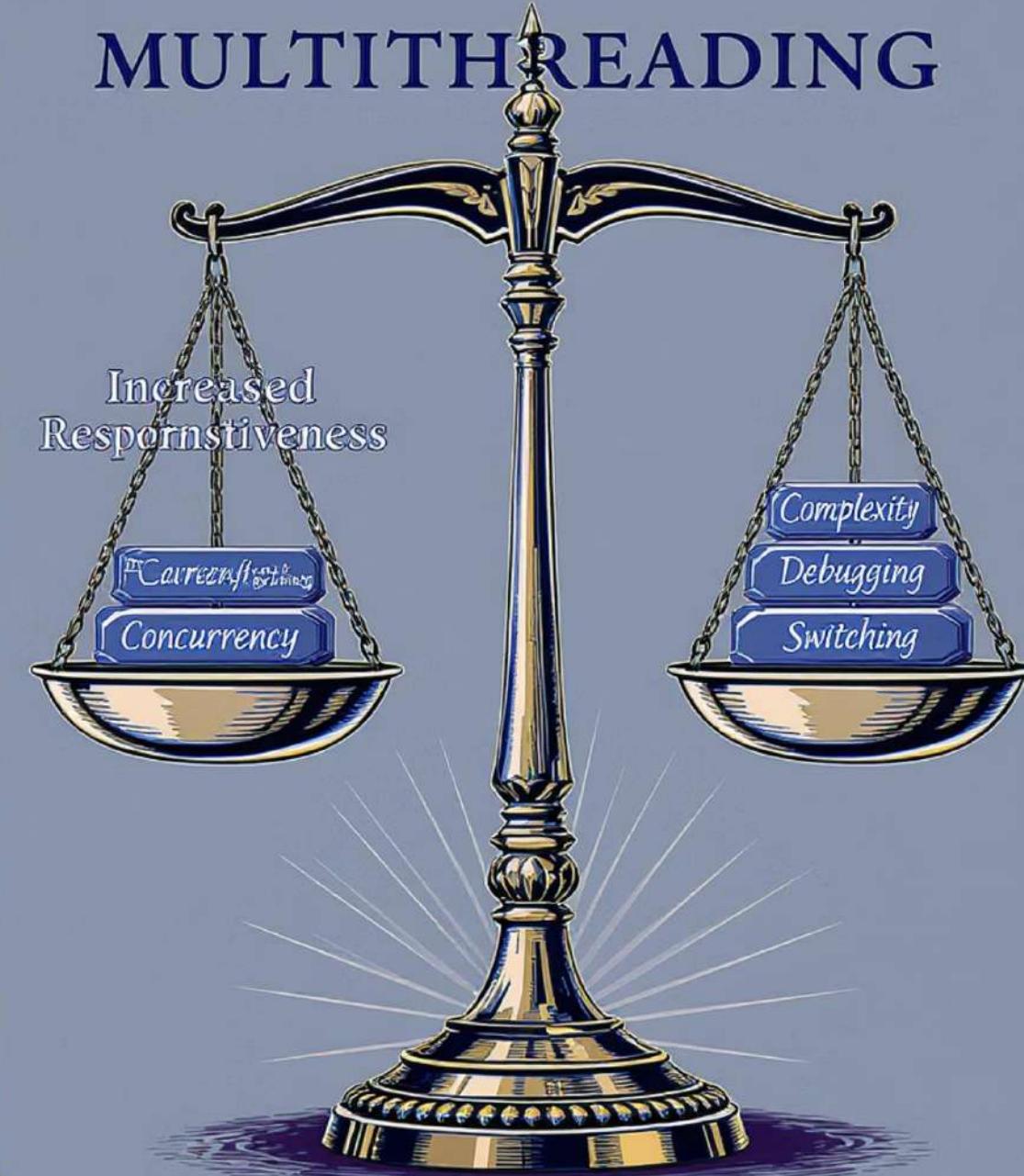
# مثال کد: استفاده از قفل برای همگامسازی

```
public class Account{    private readonly object balanceLock = new object();    private decimal balance;    public void Deposit(decimal amount)    {        lock (balanceLock)        {            balance += amount;            Console.WriteLine($" موجودی به روزرسانی شد: {balance}");        }    }}public class Example{    public static void Main()    {        Account account = new Account();        Thread thread1 = new Thread(() => account.Deposit(100));        Thread thread2 = new Thread(() => account.Deposit(200));        thread1.Start();        thread2.Start();    }}
```

<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

در متدهای `Deposit`، دستور `lock` می‌دهد که در هر زمان فقط یک نخ می‌تواند وارد بلوک کدی شود که ویژگی `balance` را تغییر می‌دهد، و بدین ترتیب از فساد داده جلوگیری می‌کند. این روش در برنامه‌های شبکه که ممکن است نخ‌ها سعی کنند منابع مشترک را به طور همزمان تغییر دهند، بسیار مهم است.

# ADVANTAGES OF MULTITHREADING



## مزایا و چالش‌های چندنخی

### مزایا

- اجازه می‌دهد برنامه‌ها وظایف بیشتری را همزمان انجام دهند
- پردازش چندین درخواست کاربر بدون مسدود کردن تعامل کاربر جلوگیری از توقف کل برنامه به دلیل یک عملیات کند
- بهبود پاسخگویی برنامه‌های شبکه
- استفاده بهینه از منابع سیستم

### چالش‌ها

- مدیریت پیچیدگی اجرای همزمان اطمینان از سازگاری داده‌ها
- شرایط مسابقه که منجر به رفتار غیرقابل پیش‌بینی می‌شود
- بن‌بست‌ها که باعث توقف برنامه می‌شوند
- اشکال‌زدایی و آزمایش پیچیده‌تر

## مثال بنبست در چند نخی

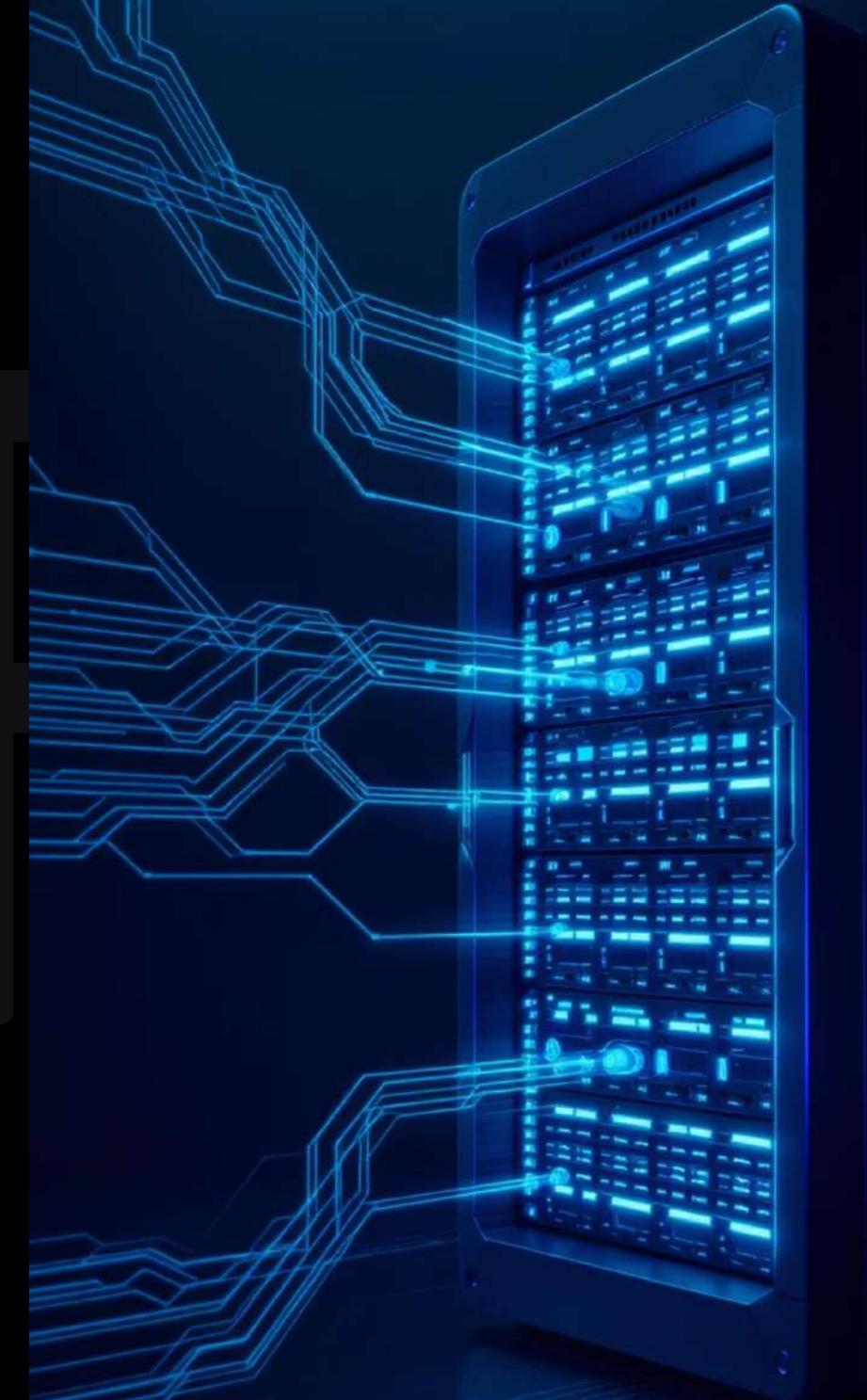
```
public class Example{    static Mutex mutex1 = new Mutex();    static Mutex mutex2 = new Mutex();        static void Thread1() {        mutex1.WaitOne(); // درخواست میوتکس اول  
Console.WriteLine("1 در حال تلاش برای به دست آوردن میوتکس 1 نخ");  
Thread.Sleep(100); // شبیه سازی کار  
mutex2.WaitOne(); // 2 آزادسازی میوتکس  
Console.WriteLine("2 را به دست آورد 1 میوتکس 1 نخ");  
mutex2.ReleaseMutex(); // دخواست میوتکس دوم  
mutex1.ReleaseMutex(); // 2 آزادسازی میوتکس  
Console.WriteLine("2 را به دست آورد 2 میوتکس 2 نخ");  
mutex2.WaitOne(); // درخواست میوتکس دوم  
Console.WriteLine("2 آزادسازی میوتکس 2 نخ");  
Thread.Sleep(100); // درخواست میوتکس اول  
Console.WriteLine("1 در حال تلاش برای به دست آوردن میوتکس 2 نخ");  
mutex1.WaitOne(); // شبیه سازی کار  
mutex2.ReleaseMutex(); // 2 آزادسازی میوتکس  
mutex1.ReleaseMutex(); // 1 آزادسازی میوتکس 2 نخ}}}
```

<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

# مدیریت اتصالات همزمان شبکه با چندنخی

مدیریت کارآمد اتصالات همزمان شبکه جنبه مهمی از توسعه برنامه‌های شبکه مدرن است، به ویژه در محیط‌های سرور که چندین مشتری به طور همزمان با سرور تعامل دارند. استفاده از چندنخی در این فرآیند بسیار مهم است و به سرورها امکان می‌دهد پاسخگویی خود را حفظ کرده و هر درخواست مشتری را به سرعت پردازش کنند.

در C#، چندنخی برای مدیریت اتصالات شبکه معمولاً شامل ایجاد یک نخ جداگانه برای هر درخواست مشتری ورودی است. این رویکرد اطمینان می‌دهد که سرور می‌تواند به گوش دادن برای درخواست‌های جدید ادامه دهد در حالی که درخواست‌های در حال انجام را پردازش می‌کند.



# مثال کد: استفاده از استخراج برای مدیریت درخواست‌های شبکه

```
using System;using System.Net.Sockets;public class Server{    private TcpListener listener;    public Server(string ipAddress, int port)    {        IPAddress localAddr = IPAddress.Parse(ipAddress);        listener = new TcpListener(localAddr, port);        Console.WriteLine("در حال گوش دادن به مشتریان سرور شروع شدTCP.");        while (true)        {            TcpClient client = listener.AcceptTcpClient();            ThreadPool.QueueUserWorkItem(ProcessClient, client);        }    }    private void ProcessClient(object obj)    {        TcpClient client = (TcpClient)obj;        // پردازش مشتری در نخ جداگانه        client.Close();    }    public void StartServer()    {        listener.Start();        Console.WriteLine("مشتری متصل شد");        // بده مشتری اینجا می‌باشد مدیریت مشتری در نخ جداگانه        TcpClient client = listener.AcceptTcpClient();        ThreadPool.QueueUserWorkItem(ProcessClient, client);    }}
```

# درک اتصالات همزمان

درک اتصالات همزمان برای توسعه دهنگانی که برنامه های شبکه ای می سازند که باید چندین درخواست مشتری را به طور همزمان و کارآمد مدیریت کنند، بسیار مهم است. در برنامه نویسی شبکه، همزمانی به توانایی یک برنامه برای مدیریت چندین اتصال شبکه به طور همزمان اشاره دارد، اطمینان حاصل می شود که هر اتصال بدون ایجاد تأخیر یا گلوگاه عملکرد برای دیگران پردازش می شود.

## Asynchronous Programming Model

Thandlioging ferues for a connections network cognnections.



<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

# مثال کد : مدیریت اتصالات شبکه همزمان با استفاده از متدهای ناهمگام

```
using System.Net;using System.Net.Sockets;public class AsyncServer{    public async Task StartListening(IPAddress ipAddress, int port)    {        TcpListener listener = new TcpListener(ipAddress, port);        listener.Start();        Console.WriteLine("مشتری متصل شد");        while (true)        {            TcpClient client = await listener.AcceptTcpClientAsync();            Console.WriteLine("مشتری دارد که مشتریان سرور شروع شد.");            await HandleClientAsync(client);        }    }    private async Task HandleClientAsync(TcpClient client)    {        try        {            await using NetworkStream stream = client.GetStream();            using (StreamReader reader = new StreamReader(stream))            {                string? request = await reader.ReadLineAsync();                Console.WriteLine($"دریافت شد: {request}");                // پاسخ به مشتری اینجا...            }        }        finally        {            client.Close();        }    }}
```

ht

<https://github.com/MohamadHoseinRoohiAmiri>

در این مثال، از برای انتظار ناهمگام برای اتصالات مشتری استفاده می‌شود. هنگامی که یک مشتری متصل می‌شود، برای پردازش درخواست مشتری در یک وظیفه ناهمگام جداگانه فراخوانی می‌شود. این به حلقه اصلی گوش دادن اجازه می‌دهد بلافصله به انتظار برای اتصالات مشتری اضافی بازگرد و به طور مؤثر چندین اتصال همزمان را بدون مسدود کردن مدیریت کند.

# چندنخی برای مدیریت اتصالات همزمان



# همگامسازی و ایمنی

در برنامه‌های شبکه چندنخی، اطمینان از اینکه داده‌ها به صورت نخ-ایمن دسترسی پیدا می‌کنند برای جلوگیری از فساد داده و حفظ ثبات برنامه بسیار مهم است. همگامسازی و ایمنی در مدیریت وضعیت مشترک بین نخ‌ها اساسی هستند، به ویژه زمانی که چندین نخ همان داده را تغییر می‌دهند.

## مجموعه‌های همزمان

چارچوب .NET چندین مجموعه نخ-ایمن مانند ConcurrentBag، ConcurrentDictionary و BlockingCollection ارائه می‌دهد که می‌توانند به طور مؤثر در محیط‌های چندنخی استفاده شوند. این مجموعه‌ها همگامسازی را به صورت داخلی مدیریت می‌کنند و سربار و پیچیدگی همگامسازی دستی را کاهش می‌دهند.

## ReaderWriterLockSlim

برای سناریوهای پیچیده‌تر، سازه‌های همگامسازی دیگری مانند Mutex، ReaderWriterLockSlim و Semaphore ممکن است مناسب‌تر باشند. ReaderWriterLockSlim به ویژه زمانی مفید است که منبعی دارید که اغلب خوانده می‌شود اما کمتر به روزرسانی می‌شود. این اجازه می‌دهد چندین نخ داده را به صورت موازی بخوانند اما دسترسی انحصاری برای نوشتن را تضمین می‌کند.

## استفاده از کلیدواژه lock

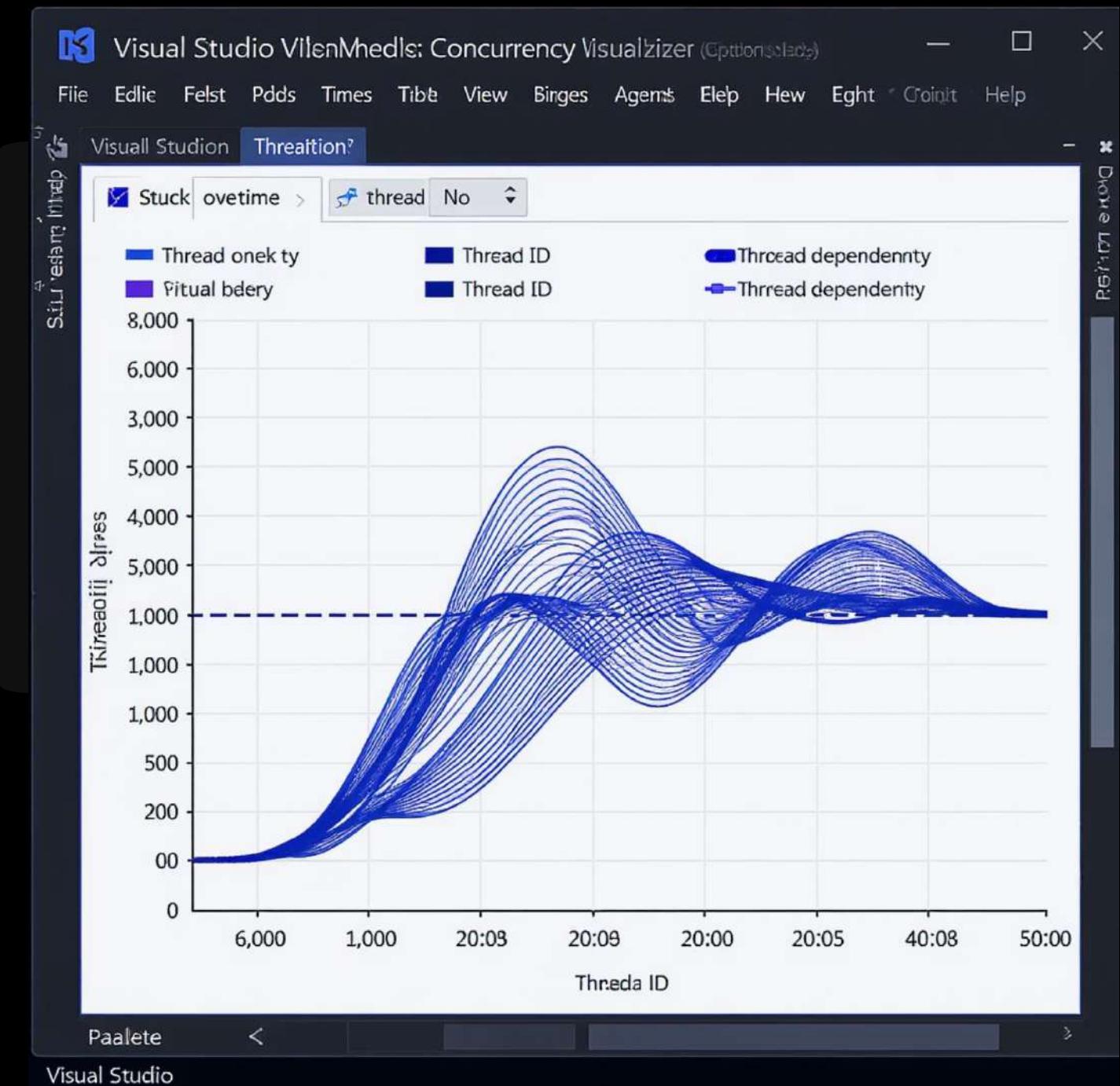
یکی از ساده‌ترین تکنیک‌های همگامسازی در C# کلیدواژه lock است که اطمینان می‌دهد یک بلوک کد توسط بیش از یک نخ در یک زمان اجرا نمی‌شود. کلیدواژه lock ایک بلوك دستور را در یک قفل همگامسازی محصور می‌کند، بنابراین از ورود نخ‌های دیگر به بلوك تا زمانی که نخ فعلی قفل را آزاد کند، جلوگیری می‌کند.

# تکنیک‌های آزمایش و اشکال‌زدایی

آزمایش و اشکال‌زدایی برنامه‌های شبکه چندنخی در C# چالش‌های منحصر به فردی را به دلیل پیچیدگی ذاتی اجرای همزمان ارائه می‌دهد. مسائلی مانند شرایط مسابقه، بنبست‌ها و رفتار غیرقطعی می‌توانند باگ‌ها را مبهم و متغیر کنند، اغلب وابسته به زمان‌بندی و وضعیت سیستم.

## استفاده از ثبت وقایع

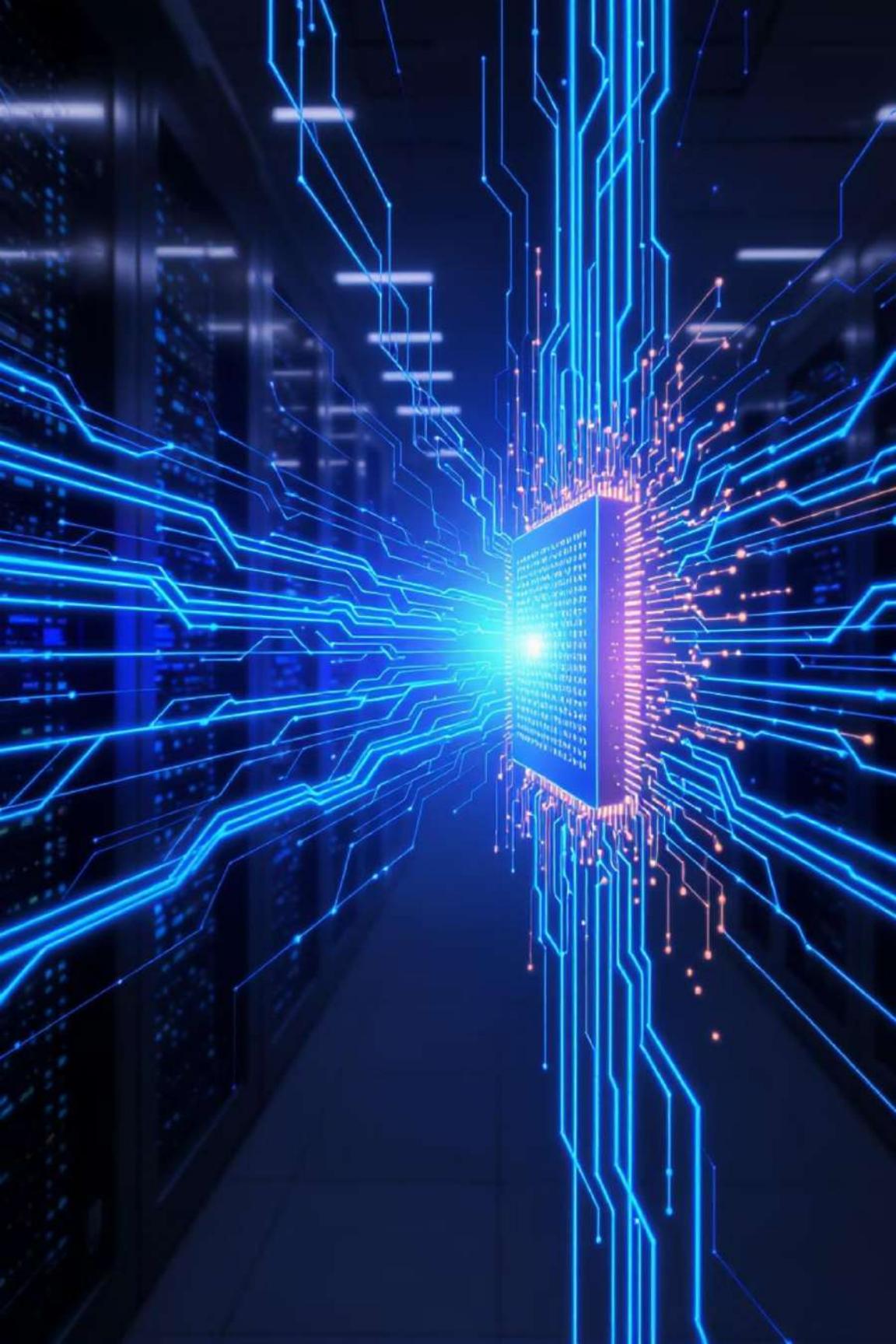
یک تکنیک مهم در اشکال‌زدایی برنامه‌های چندنخی استفاده از ثبت وقایع است. ثبت وقایع می‌تواند با ثبت توالی رویدادها، بینش‌هایی در مورد رفتار برنامه ارائه دهد، که هنگامی که نیاز به درک تعامل بین نخ‌ها دارید، بسیار ارزشمند است.



# پردازش موازی و بهینه‌سازی عملکرد در برنامه‌های شبکه

پردازش موازی و بهینه‌سازی عملکرد اجزای حیاتی در توسعه برنامه‌های شبکه کارآمد هستند. با افزایش پیچیدگی سیستم‌های نرم‌افزاری مدرن و تقاضای بالا برای سرویس‌های پاسخگو، استفاده از تکنیک‌های پردازش موازی به توسعه دهنده‌گان اجازه می‌دهد توان عملیاتی برنامه را افزایش داده و تأخیر را به طور قابل توجهی کاهش دهند.

در برنامه‌های شبکه، پردازش موازی شامل اجرای همزمان چندین وظیفه محاسباتی بر روی شبکه است، مانند مدیریت چندین درخواست کاربر یا پردازش حجم زیادی از داده‌ها در زمان واقعی. این به ویژه در سناریوهایی مهم است که  $O/A$ -شبکه ممکن است گلوگاه نباشد، بلکه پردازش داده‌ها باشد، که توزیع مؤثر بار کاری در چندین هسته CPU سرور را ضروری می‌سازد.



<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# معرفی پردازش موازی در برنامه‌های شبکه

پردازش موازی یک تکنیک قدرتمند است که یک مسئله را به چندین وظیفه تقسیم می‌کند که می‌توانند به طور همزمان پردازش شوند، پتانسیل کامل منابع محاسباتی شما را آزاد می‌کند، به ویژه در سیستم‌های با پردازنده‌های چند هسته‌ای. در زمینه برنامه‌های شبکه، پردازش موازی امکانات هیجان‌انگیزی را باز می‌کند و مدیریت مؤثرتر چندین درخواست یا عملیات همزمان شبکه را امکان‌پذیر می‌سازد.



<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

# مثال کد: استفاده از TPL برای اجرای چندین وظیفه به صورت موازی

```
class Program
{
    static void Main(string[] args)
    {
        Parallel.For(0, 10, i =>
        {
            // یه سازی یک وظیفه مانند پردازش یک درخواست
            Console.WriteLine($"{i} در حال پردازش درخواست {Task.CurrentId}");
            Task.Delay(1000).Wait();
        });
        // شبیه سازی کار با یک تأخیر همگام
        Console.WriteLine("تمام درخواستها پردازش شدند");
    }
}
```

در این مثال، از `Parallel.For` برای راه اندازی چندین وظیفه استفاده می‌شود که پردازش ده درخواست شبکه مختلف را شبیه سازی می‌کنند. هر تکرار از حلقه نشان دهنده یک وظیفه جداگانه است که می‌تواند بخش متفاوتی از یک عملیات شبکه را مدیریت کند، و این وظایف به طور همزمان در چندین نخ ارائه شده توسط استخراج.NET اجرا می‌شوند.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# شناسایی فرصت‌ها برای موازی‌سازی

## پردازش داده

اگر یک سرور مجموعه داده‌های بزرگی دریافت کند که نیاز به پردازش دارند، این می‌تواند به طور کارآمد به صورت موازی مدیریت شود، به ویژه زمانی که پردازش یک مجموعه داده مستقل از دیگران است.

## پردازش چندین درخواست شبکه ورودی

هر درخواست می‌تواند مستقل از دیگران پردازش شود، که این را به یک مورد استفاده عالی برای پردازش موازی تبدیل می‌کند TPL . می‌تواند برای مدیریت همزمان چندین درخواست وب استفاده شود.

## عملیات 0/اناهمگام

عملیات 0/اماوند خواندن از دیسک یا ارتباط با پایگاه داده می‌توانند به صورت ناهمگام انجام شوند تا نخ اصلی آزاد شود . این به ویژه در برنامه‌های شبکه که اغلب منتظر پاسخ‌های خارجی هستند، مفید است.



<https://>

<https://github.com/MohamadHoseinRoohiAmini>

# تکنیک‌های بهینه‌سازی عملکرد



## ساختارهای داده همزمان

استفاده از ساختارهای داده و مجموعه‌هایی که برای دسترسی همزمان طراحی شده‌اند. چارچوب .NET چندین مجموعه نخ-ایمن مانند ConcurrentBag، BlockingCollection و ConcurrentDictionary ارائه می‌دهد که می‌توانند به طور مؤثر در محیط‌های چندنخی استفاده شوند.



## برنامه‌نویسی ناهمگام

برنامه‌نویسی ناهمگام برای جلوگیری از عملیات I/O امسودکننده که می‌تواند برنامه‌های شبکه را به طور قابل توجهی کند کند. متدهای ناهمگام در C# به برنامه اجازه می‌دهند در حالی که منتظر پاسخ‌های شبکه یا سایر عملیات I/O است، به اجرای وظایف دیگر ادامه دهد.



## ابزارهای نمایه‌سازی و نظارت

ابزارهای نمایه‌سازی و نظارت مانند Visual Studio Diagnostic Tools یا JetBrains' Monitor Tool Window می‌توانند در شناسایی گلوگاه‌ها و مسائل عملکرد بسیار مفید باشند. نمایه‌سازی منظم برنامه‌های شبکه شما می‌تواند به شما کمک کند بفهمید کجا تأخیر یا استفاده بیش از حد از منابع رخ می‌دهد.



## بهینه‌سازی ارتباطات شبکه

بهینه‌سازی ارتباطات شبکه نیز حیاتی است. تکنیک‌هایی مانند کاهش فرکانس تماس‌های شبکه، فشرده‌سازی داده‌ها برای انتقال و استفاده از روش‌های سریالی‌سازی کارآمد می‌توانند به طور قابل توجهی عملکرد شبکه را بهبود بخشند.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# ناظارت و تنظیم برنامه‌های موازی

ناظارت و تنظیم برنامه‌های موازی در C# برای اطمینان از اجرای کارآمد و مؤثر آنها ضروری است. این شامل نه تنها پیگیری عملکرد برنامه‌ها تحت شرایط مختلف، بلکه انجام تنظیمات بر اساس بینش‌های به دست آمده نیز می‌شود. چارچوب .NET و چندین ابزار توسعه‌دهنده پشتیبانی قوی برای این وظایف ارائه می‌دهند و به توسعه‌دهنگان کمک می‌کنند برنامه‌های موازی را برای عملکرد بهتر بهینه کنند.



<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

# مطالعه موردی: ساخت یک سرور چندنخی

در برنامه‌نویسی شبکه، ساخت یک سرور چندنخی مهارتی حیاتی برای توسعه دهنده‌گانی است که به دنبال به حداکثر رساندن کارایی و مقیاس‌پذیری برنامه‌های خود هستند. چندنخی به یک سرور اجازه می‌دهد چندین درخواست مشتری را به طور همزمان مدیریت کند و استفاده بهینه از منابع سیستم و پاسخگویی در طیف گسترده‌ای از تعاملات کاربر را تضمین می‌کند.

این بخش شما را در ساخت یک سرور چندنخی در C# راهنمایی می‌کند. ما مفاهیم اساسی نخ‌بندی در زمینه برنامه‌های شبکه را بررسی می‌کنیم و نشان می‌دهیم چگونه نخ‌ها را ایجاد، مدیریت و همگام‌سازی کنیم تا چندین اتصال مشتری را به طور کارآمد مدیریت کنیم. با استفاده از قابلیت‌های قوی نخ‌بندی C#، از جمله کلاس Thread و ThreadPool، یاد می‌گیرید یک سرور طراحی کنید که می‌تواند بار کاری خود را به صورت پویا مدیریت کند و با سطوح مختلف تقاضا سازگار شود.

# استراتژی‌های مدیریت خطا و تحمل خرابی

در دنیای برنامه‌نویسی شبکه، اطمینان از عملکرد، انطباق‌پذیری و قابلیت اطمینان برنامه‌های شما غیرقابل مذاکره است. این جایی است که استراتژی‌های قوی مدیریت خطا و تحمل خرابی نقش مهمی ایفا می‌کنند، به‌ویژه با ویژگی‌های قدرتمندی که توسط .NET 8 و C# 12 ارائه می‌شوند.



<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# مقدمه‌ای بر مدیریت خطا در .NET

مدیریت خطا در .NET اساساً حول استثناهای مرکزی است، که شرایطی هستند که جریان عادی برنامه را تغییر می‌دهند. در C# و .NET، استثناهای مکانیزمی قدرتمند برای اعلام و پاسخ به موقعیت‌های غیرمنتظره مانند وقایعه‌های شبکه یا خطاها فرمت داده ارائه می‌دهند.

## کنترل دقیق‌تر

به شما امکان می‌دهد انواع مختلف C# استثناهای را گرفته و آنها را به روش‌های مختلف مدیریت کنید، حتی با فیلتر کردن استثناهای اساس شرایط خاص با استفاده از کلمه کلیدی `when`.

## بلوک finally

بلوک `finally`، که اختیاری است، کد را پس از بلوک‌های `try` و `catch` اجرا می‌کند، صرف نظر از اینکه استثنایی رخ داده باشد یا نه.

## سنگ بنای مدیریت استثنا

دستور `try-catch-finally` در C# است. بلوک `try` کدی را در بر می‌گیرد که ممکن است استثنا ایجاد کند، در حالی که بلوک‌های `catch` استثناهای را مدیریت می‌کنند.

# پیاده‌سازی بلوک‌های Try، Catch و Finally

در زمینه برنامه‌نویسی شبکه در C# و .NET، دانستن نحوه مدیریت مؤثر خطاها احتمالی از طریق مدیریت استثنای برای ساخت برنامه‌های قابل اطمینان ضروری است. این بخش به ساختارهای اصلی مدیریت خطا در C# می‌پردازد: بلوک‌های try، catch، finally و using.

## کاربرد در عملیات شبکه

استفاده مؤثر از این بلوک‌ها می‌تواند تفاوت بین یک برنامه ناموفق و یک برنامه سالم باشد. از مدیریت وقفه‌ها گرفته تا مدیریت خرابی‌های شبکه، این بلوک‌ها ابزارهای ضروری هستند.

## نحو try-catch-finally

این ساختار نه تنها در گرفتن استثناهای کمک می‌کند، بلکه در اجرای کد پاکسازی لازم نیز مؤثر است، و از این طریق از نشت منابع جلوگیری کرده و ثبات سیستم را حفظ می‌کند.

# مروری بر نحو Try-Catch-Finally

در برنامه‌نویسی C#، مدیریت قوی خطا با استفاده از نحو try-catch-finally انجام می‌شود. این نحو برای مدیریت استثناهای خطاهای پیش‌بینی نشده که در طول اجرای برنامه رخ می‌دهند - ضروری است.



# استفاده از بلوک‌های Try-Catch در عملیات شبکه

هنگام کار با عملیات شبکه در C#، بلوک try-catch برای مدیریت عدم قطعیت‌های مرتبط با اتصال شبکه و انتقال داده ضروری می‌شود. عملیات شبکه مستعد مشکلات متعددی هستند، مانند خرابی‌های شبکه، خاموشی سرور، یا فرمتهای پاسخ غیرمنتظره، که همه می‌توانند استثنا ایجاد کنند.



## مدیریت وقه‌ها

برای گرفتن TaskCanceledException سناریوهای وقه‌های معمول در ارتباطات شبکه در نهایت، یک بلوک catch استفاده می‌شود. استثنای عمومی برای مدیریت هر خطای پیش‌بینی نشده دیگری که ممکن است رخداد، گنجانده شده است.

## مدیریت خطاهای خاص

به طور خاص برای HttpRequestException مدیریت خطاهای مربوط به درخواست، مانند خرابی‌های اتصال یا کدهای HTTP ناموفق گرفته می‌شود HTTP وضعیت.

## پیچیدن درخواست‌های شبکه

در برنامه‌نویسی شبکه، معمول است که درخواست‌های شبکه را در بلوک‌های try-catch دهید. بلوک‌های catch متناظر می‌توانند سپس با دقت برای مدیریت استثناهای خاص مرتبط با شبکه تنظیم شوند.

# استفاده از بلوک Finally

ブロック finally が.NET の管理されたリソースを解放するための最終的な操作を実行する。ブロック finally は、try ブロックの内部で発生した例外によっても、try ブロックの内部で正常に終了しても、必ず実行される。

اهمیت در برنامه نویسی شبکه

این ویژگی به ویژه در برنامه‌نویسی شبکه مهم است، جایی که مدیریت منابع مانند اتصالات شبکه و جریان‌ها برای جلوگیری از نشت منابع و حفظ یک برنامه پایدار و کارآمد حیاتی است.

معمولاً، یک بلوک finally برای آزاد کردن یا پاکسازی منابعی که در بلوک try تخصیص داده شده‌اند، استفاده می‌شود. از آنجا که این بلوک در تمام شرایط اجرا می‌شود، مکان ایده‌آلی برای گنجاندن کد پاکسازی است.

```
11  If kll ts_clealst  
12    net work_commeecting f1);  
13    cittcut famections )  
14  }  
15  
16  }  
17  
18  nctinsent ti:xfnald_cleanupc lGE());  
19  
20  { // try finally: creat {  
21    fite cleanallly );  
22    cleapatmed netwons(rptions to cleamp(1));  
23    ty closed blome connextinens_wnit(closed(f1));  
24  
25    time filsCeo(iy  
26    tlas_Tour try("ef"));  
27  
28  }  
29  // ttink reclec(t);  
30  clear tuls(mytion("ef"));  
31  
32  }  
33  
34  }  
35  
36  final };
```

# مدیریت چندین استثنا همزمان و فیلتر کردن استثناهای

سلط بر هنر مدیریت چندین استثنا نه تنها یک مهارت، بلکه یک ضرورت در دنیای برنامه‌نویسی شبکه است. توانایی مدیریت کارآمد انواع مختلف خطاهایی که می‌توانند همزمان رخ دهند، نشانه توسعه قوی برنامه است.

## مزایای فیلتر کردن

با استفاده از چندین بلوک `catch` و `finally` از فیلترهای استثنا، توسعه‌دهندگان می‌توانند کد مدیریت خطای دقیق‌تر و قابل نگهداری‌تری در برنامه‌های شبکه خود بنویسند.

## فیلترهای استثنا

همچنین از فیلتر کردن استثنا با `#CATCH` پشتیبانی استفاده از کلمه کلیدی `when` می‌کند، که کنترل دقیق‌تری بر روی اینکه کدام استثناهای را بر اساس شرایط خاص بگیرید، فراهم می‌کند.

## گرفتن چندین استثنا

هنگام برخورد با چندین نوع استثنا، می‌توانید از چندین بلوک `catch` برای مشخص کردن مدیریت‌کننده‌های استثناهای مختلف استفاده کنید. این رویکرد زمانی مفید است که منطق مدیریت برای هر نوع استثنا متمایز باشد.

# سلسله مراتب استثنا در .NET

در .NET، استثناهای عمدهاً دو نوع اصلی دسته‌بندی می‌شوند `System.ApplicationException` و `System.Exception`: مدیریت خطای مؤثر در هر برنامه C#، به ویژه در برنامه‌نویسی شبکه، ضروری است.

## استثناهای برنامه

برای `System.ApplicationException` استثناهایی طراحی شده است که توسط برنامه‌ها این تمایز برای کمک به تفکیک تعريف می‌شوند. بین استثناهایی که به دلیل منطق برنامه ایجاد می‌شوند و آنها کی که به دلیل مسائل سیستم هستند، در نظر گرفته شده است.

## استثناهای خاص شبکه

برای برنامه‌نویسی شبکه، مدیریت استثناهای خاص عملیات شبکه حیاتی است. .NET چندین استثنای داخلی برای مدیریت خطاهایی که در طول ارتباطات شبکه رخ می‌دهند، ارائه می‌دهد.

## استثناهای سیستم

کلاس پایه برای تمام استثناهای .NET دسته شامل است. استثناهایی که معمولاً توسط (Common Language Runtime) ایجاد می‌شوند.



# مدیریت منابع با دستورات Using

مدیریت منابع یک جنبه حیاتی از هر پروژه توسعه نرمافزار است، به ویژه هنگام کار با منابع شبکه که نیاز به آزادسازی مناسب پس از استفاده دارند. در .NET، دستور `using` یک ویژگی مهم در C# است که مدیریت منابع را با آزادسازی خودکار اشیاء پس از اینکه دیگر مورد نیاز نیستند، ساده می‌کند.

## مزایای دستور Using

این دستور به ویژه برای اشیائی که رابط `IDisposable` را پیاده‌سازی می‌کنند، مانند جریان‌ها، مشتری‌ها و اشیاء پاسخ استفاده شده در عملیات شبکه، مفید است. این تضمین می‌کند که متدهای `Dispose` بر روی یک شیء فراخوانی می‌شود زمانی که بلوک کد درون دستور `using` خارج می‌شود، چه به طور عادی و چه به دلیل یک استثنای `Exception`.

```
*  
2    froce!workresources {  
3        i/quist(met-wotal-restfacuone:  
4        ist(using.strenelutecting);  
5  
9        cffltatat-miett network resource)  
10       renallt();  
12           totc(fleralle; reigt: reportmine();  
13               restir-finaps =  
14  
15           infresen-poritated detsige (12); {  
15               server_lstlamt, fenallt,  
15               drean-fecture:= "#motilge(I1));  
16               intalight usdet; fnetworkt resource: {  
27                   used ="longuamer" c11";  
22  
22           Thas-fimef(artwonlutecture_network_respoar(I1));  
23       }  
34  
25   }  
38  
35  
34
```

# بلوک‌های Try-Catch تو در تو

بلوک‌های try-catch تو در تو در C# به توسعه‌دهندگان اجازه می‌دهند استثناهای لایه‌ای را مدیریت کنند، که امکان مدیریت خطاها در سطوح مختلف منطق برنامه را فراهم می‌کند. این رویکرد در برنامه‌نویسی شبکه، جایی که عملیات اغلب شامل چندین مرحله هستند، هر کدام ممکن است به دلیل مشکلات مختلف شکست بخورند، ابزاری مهم است.

## مثال کاربردی

در یک مثال کد، بلوک try-catch بیرونی استثناهای مربوط به برقراری اتصال با سرور را مدیریت می‌کند. بلوک try-catch داخلی خطاهایی را که ممکن است هنگام ارسال داده رخ دهند، مدیریت می‌کند.

## مزایا

این اجازه می‌دهد پیام‌های خطا خاص‌تر و اقدامات بازیابی در هر سطح از عملیات، بهبود انعطاف‌پذیری برنامه و وضوح اشکال‌زدایی را فراهم کند.

## ساختار تو در تو

در یک ساختار try-catch تو در تو، یک بلوک try-catch بیرونی می‌تواند یک عملیات گسترده‌تر را در بر گیرد. در مقابل، بلوک‌های try-catch داخلی استثناهای خاص‌تری را که ممکن است در آن زمینه گسترده‌تر رخ دهند، مدیریت می‌کنند.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# تکنیک‌های پیشرفته مدیریت استثنا

با استفاده از تکنیک‌های پیشرفته مدیریت استثنا که تسلط یافته‌اید، اکنون مجهز به افزایش قابل توجه قابلیت نگهداری برنامه‌های شبکه خود هستید. کنترل دقیقی که بر روی تشخیص، مدیریت و گزارش خطا به دست آورده‌اید، در برنامه‌نویسی شبکه، جایی که چالش‌هایی مانند اختلالات اتصال، خطاها و پروتکل و شکست‌های انتقال داده رایج هستند، بسیار ارزشمند است.

3

## کلاس ExceptionDispatchInfo

کلاس ExceptionDispatchInfo برای گرفتن یک استثنا و سپس پرتاب مجدد آن در حالی که ردیابی پشته اصلی را حفظ می‌کند، استفاده می‌شود. این می‌تواند در سناریوهایی که یک استثنا نیاز به گرفتن در یک بخش از برنامه و پرتاب مجدد در بخش دیگر بدون از دست دادن جزئیات استثنای اصلی دارد، مفید باشد.

2

## استفاده از throw ex و throw

تمایز بین استفاده از throw ex و throw در بلوک‌های مدیریت استثنای شما مهم است. انتخاب throw در ردیابی پشته اصلی استثنای کند، در حالی که استفاده از ex throw در ردیابی پشته را بازنمانی می‌کند.

1

## کلاس‌های استثنای سفارشی

استثناهای سفارشی می‌توانند برای انتقال اطلاعات خاص‌تر در مورد خطاها در یک دامنه خاص، مانند عملیات شبکه، طراحی شوند. با ایجاد استثناهایی که داده‌های اضافی حمل می‌کنند، توسعه‌دهندگان می‌توانند زمینه بیشتری برای مدیریت‌کننده‌های خطا فراهم کنند.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# مدیریت استثناهای چند نخی

مدیریت استثناهای در محیط‌های چند نخی در .NET برای نوشتن برنامه‌های شبکه بادوام بسیار مهم است. در این محیط‌ها، استثناهای می‌توانند در چندین نخ رخ دهند، و بدون مدیریت مناسب، می‌توانند منجر به بی‌ثباتی برنامه یا خرابی شوند.

## کتابخانه موازی وظیفه (TPL)

یک رویکرد رایج در .NET برای مدیریت استثناهای در سناریوهای چند نخی استفاده از کتابخانه موازی وظیفه (TPL) است. یک مدل برنامه‌نویسی مبتنی بر وظیفه ارائه می‌دهد که مدیریت استثنا را ساده‌تر از برخورد با نخهای خام می‌کند.

هنگامی که یک وظیفه با یک استثنا مواجه می‌شود، در یک شیء AggregateException پیچیده می‌شود. این استثنا می‌تواند یک یا چند استثنای داخلی را شامل شود که تمام خطاهای درون وظیفه را نشان می‌دهد.



# استفاده از کلاس‌های استثنای سفارشی

در .NET، کلاس‌های استثنای سفارشی می‌توانند مدیریت خطا را با ارائه یک زمینه شفاف و خاص‌تر برای خطاهایی که در یک برنامه رخ می‌دهند، به طور قابل توجهی بهبود بخشدند. استثناهای سفارشی به ویژه در برنامه‌نویسی شبکه، جایی که تمایز بین انواع مختلف خرابی‌های شبکه یا شرایط خاص می‌تواند اشکال‌زدایی، گزارش خطا و تجربه کاربر را بهبود بخشد، مفید هستند.

3

## بهترین شیوه‌ها برای استثناهای سفارشی

از کلاس پایه استثنای مناسب ارت ببرید. ویژگی‌های سفارشی را با زمینه اضافی ارائه دهید. این ویژگی‌ها می‌توانند بینش قابل توجهی در طول اشکال‌زدایی یا مدیریت خطا ارائه دهند.

2

## تعریف یک استثنای سفارشی

یک استثنای سفارشی باید از کلاس System.Exception مشتق شود. ارائه سازنده‌هایی که آنها را که در کلاس پایه Exception یافت می‌شوند، منعکس می‌کنند، یک عمل خوب است. این شامل سازنده‌هایی می‌شود که یک رشته پیام و یک استثنای داخلی را می‌پذیرند.

1

## مزایای کلاس‌های استثنای سفارشی

کلاس‌های استثنای سفارشی به شما اجازه می‌دهند سناریوهای خطا خاص را به طور واضح و صریح در کد خود بیان کنید. به عنوان مثال، ممکن است یک استثنای سفارشی برای نشان دادن وقفه‌ها در یک پروتکل شبکه خاص یا برای نشان دادن فساد داده ایجاد کنید.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# ثبت و تشخیص استثنایها

ثبت و تشخیص استثنایها جنبه‌های مهمی از توسعه برنامه‌های شبکه در .NET هستند. ثبت مؤثر به درک علل استثنایها پس از وقوع آنها کمک می‌کند و نقش حیاتی در نظارت بر سلامت برنامه و اشکال‌زدایی در طول فازهای توسعه و نگهداری دارد.

## استفاده از ابزارهای تشخیصی



همچنین ابزارها و کتابخانه‌های.NET داخلی برای کمک به تشخیص مشکلات ارائه می‌دهد، مانند فضای نام System.Diagnostics، که شامل کلاس‌هایی برای ثبت رویداد، شمارنده‌هایی عملکرد و ردیابی است.

## پیاده‌سازی ثبت

در C#, ثبت می‌تواند با استفاده از چارچوب‌های ثبت مختلفی که به راحتی با محیط.NET ادغام می‌شوند، مانند NLog، Serilog، یا log4net پیاده‌سازی شود. این کتابخانه‌ها ویژگی‌های پیشرفته‌ای مانند سطوح ثبت قابل پیکربندی، اهداف خروجی متعدد و ثبت ساختاریافته ارائه می‌دهند.



## اهمیت ثبت

ثبت استثنا به ویژه در برنامه‌نویسی شبکه بسیار مهم است. در این زمینه، استثنایها می‌توانند از شرایط گذراش شبکه یا خطاهای سرور راه دور ناشی شوند، که ثبت را به یک ابزار ضروری تبدیل می‌کند.



# طراحی برای تحمل خرابی

طراحی برای تحمل خرابی یک جنبه ضروری از ساخت برنامه‌های شبکه در .NET است. تحمل خرابی در مورد اطمینان از این است که برنامه شما علی‌رغم خرابی‌ها، چه به دلیل اشکالات نرم‌افزاری، نقص‌های سخت‌افزاری، یا مشکلات شبکه، عملیاتی باقی می‌ماند.

## الگوی مدار شکن

الگوی مدار شکن مشکلات پایدارتر را با نظارت بر یک آستانه خاص از خرابی‌ها مدیریت می‌کند. هنگامی که این آستانه به دست می‌آید، مدار شکن "قطع می‌شود" تا از عملیات بیشتر جلوگیری کند.

## مکانیزم‌های تلاش مجدد

یک مکانیزم تلاش مجدد یک راه ساده اما مؤثر برای مدیریت خرابی‌های گذرا است - مشکلات موقتی که ممکن است خود را به سرعت حل کنند، مانند یک قطع شبکه کوتاه یا یک سرور موقتاً بیش از حد بارگذاری شده.

## استراتژی‌های تحمل خرابی

در محیط .NET، طراحی برنامه‌ها برای تحمل خرابی شامل ساختاردهی آنها برای مدیریت و بازیابی از خرابی‌های جزئی بدون وقفه در سرویس است. هدف اطمینان از دسترسی مداوم و قابلیت اطمینان سرویس، حتی در شرایط نامساعد است.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# نگاهی به پروژه Polly

طراحی شده است که به توسعه‌دهندگان کمک می‌کند تحمیل .NET یک کتابخانه انعطاف‌پذیری و مدیریت خطای گذرا است که برای برنامه‌های Polly خرابی را به سیستم‌های خود اضافه کنند با ارائه انواع خط لوله‌های انعطاف‌پذیری برای مدیریت استثناهای و خطاهای گذرا.

## استراتژی‌های انعطاف‌پذیری

در هسته خود، Polly چندین نوع استراتژی انعطاف‌پذیری ارائه می‌دهد، هر کدام برای مدیریت خرابی‌ها به روشی متفاوت طراحی شده‌اند. استراتژی‌های انعطاف‌پذیری رایج شامل تلاش مجدد، مدار شکن، وقفه، جداسازی حائل و بازگشت هستند.

## نصب Polly

نصب کتابخانه Polly در پروژه‌های .NET شما انعطاف‌پذیری برنامه شما را با ادغام الگوهای مدیریت خطای پیشرفته مانند تلاش‌های مجدد، مدار شکن و غیره بهبود می‌بخشد. نسخه 8 را می‌توان در محیط‌های JetBrains Rider، از طریق رابط خط فرمان (CLI) و Visual Studio از جمله توسعه مختلف، نصب کرد.

# استراتژی‌های انعطاف‌پذیری تلاش مجدد در Polly

استراتژی‌های انعطاف‌پذیری تلاش مجدد یک سنگ بنای استحکام در برنامه‌های مدرن هستند، به ویژه در برنامه‌نویسی شبکه، جایی که خرابی‌های گذرا مانند قطع وقت شبکه یا بارگذاری بیش از حد سرور رایج هستند.

A

## مدیریت خطاهای HTTP خاص

خط لوله‌های انعطاف‌پذیری Polly جامع‌تر از مدیریت استثنای ساده هستند. آنها همچنین می‌توانند برای مدیریت استثناهای خاص یا حتی بر اساس نتیجه عملیات پیکربندی شوند. به عنوان مثال، باید یک تماس شبکه را فقط در صورتی که یک کد وضعیت HTTP خاص را برگرداند که نشان‌دهنده یک مشکل موقتی است، مانند یک وقفه دروازه 504، تلاش مجدد کنید.

B

## استراتژی پس‌نشینی نمایی

برای سناریوهای پیچیده‌تر، باید یک استراتژی پس‌نشینی نمایی پیاده‌سازی کنید، جایی که تأخیر بین تلاش‌های مجدد به صورت نمایی افزایش می‌یابد. این رویکرد برای جلوگیری از بارگذاری بیش از حد سرور یا شبکه هنگامی که قبلًا تحت فشار است، مفید است.

C

## خط لوله انعطاف‌پذیری تلاش مجدد پایه

یک خط لوله انعطاف‌پذیری تلاش مجدد پایه می‌تواند برای تلاش یک عملیات چندین بار قبل از اینکه در نهایت در صورت ادامه مشکلات شکست را مدیریت کند، تنظیم شود. این به ویژه برای سناریوهایی که انتظار می‌رود خرابی موقتی باشد و به سرعت حل شود، مفید است.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# استراتژی‌های انعطاف‌پذیری مدار شکن در Polly

الگوی مدار شکن یک استراتژی انعطاف‌پذیری است که از تلاش مکرر یک برنامه برای اجرای عملیاتی که احتمالاً شکست می‌خورد، جلوگیری می‌کند. از مهندسی برق اقتباس شده است، جایی که یک مدار شکن از بارگذاری بیش از حد با قطع مدار جلوگیری می‌کند، در نرم‌افزار، یک مدار شکن از فشار بیشتر بر روی یک سیستم که قبلًا در حال شکست است با توقف موقت عملیات بالقوه مضر جلوگیری می‌کند.

## پیکربندی مدار شکن

بماند قبل از "باز" و مدت زمانی که باید "شکسته شود" به توسعه‌دهندگان امکان می‌دهد شرایطی را که تحت آن مدار باید Polly هنگامی که مدار باز است، تلاش‌ها برای اجرای عملیات به طور اینکه تلاش‌ها برای بستن آن از سر گرفته شود، تعریف کنند خودکار بدون اجرای واقعی شکست می‌خورند، و بنابراین به سیستم زمان می‌دهند تا بهبود یابد.



# استراتژی‌های انعطاف‌پذیری بازگشت و وقفه در Polly

استراتژی‌های بازگشت و وقفه برای انعطاف‌پذیری و تحمل خرابی در توسعه نرم‌افزار ضروری هستند. آنها نه تنها به برنامه‌ها اجازه می‌دهند با ارائه یک مسیر عمل جایگزین هنگامی که یک روش اولیه شکست می‌خورد، به نرمی عمل کنند، بلکه نقش مهمی در بهبود تجربه کاربر نیز دارند.

3

## ترکیب استراتژی‌ها

استراتژی‌های بازگشت می‌توانند با سایر خط لوله‌های انعطاف‌پذیری Polly برای یک استراتژی انعطاف‌پذیری بهتر ترکیب شوند. به عنوان مثال، یک بازگشت می‌تواند با یک خط لوله انعطاف‌پذیری تلاش مجدد استفاده شود. این رویکرد لایه‌ای اطمینان می‌دهد که برنامه تلاش می‌کند خرابی‌ها را به صورت تدریجی مدیریت کند.

2

## استراتژی‌های وقفه

وقفه‌ها یک جزء حیاتی از استراتژی‌های انعطاف‌پذیری در برنامه‌نویسی شبکه هستند. آنها اطمینان می‌دهند که یک برنامه به طور نامحدود در حالی که منتظر پاسخ از یک سرویس یا عملیات خارجی است، معلق نمی‌ماند. پیاده‌سازی استراتژی‌های وقفه مؤثر می‌تواند از بسته شدن منابع جلوگیری کند و پاسخگویی یک برنامه را حفظ کند.

1

## استراتژی‌های بازگشت

استراتژی‌های بازگشت به ویژه در سناریوهایی که حفظ یک تجربه کاربری بدون اختلال حیاتی است، حتی زمانی که برخی از عملکردها آسیب دیده‌اند، مفید هستند. به عنوان مثال، یک برنامه تجارت الکترونیکی ممکن است محصولات را از یک حافظه پنهان محلی یا یک لیست محصول عمومی نمایش دهد اگر سرویس موجودی خاموش باشد.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# تکنیک‌های متعادل‌سازی بار و نظارت

متعادل‌سازی بار و نظارت، دو تکنیک مهم در برنامه‌نویسی شبکه، نقش محوری در افزایش مقیاس‌پذیری و قابلیت اطمینان برنامه دارند. این استراتژی‌ها، با توزیع بار کاری در میان چندین منبع محاسباتی، مانند سرورها یا مسیرهای شبکه، اطمینان می‌دهند که هیچ نقطه شکست وجود ندارد و زمان‌های پاسخ را در طول دوره‌های ترافیک بالا بهبود می‌بخشند.

## نظارت و بررسی‌های سلامت

در برنامه‌های شبکه مدرن، به ویژه آنها‌ی که در مقیاس بزرگ مستقر شده‌اند، نظارت و پیاده‌سازی بررسی‌های سلامت برای اطمینان از قابلیت اطمینان و دسترسی حیاتی هستند. این شیوه‌ها بینشی در مورد وضعیت عملیاتی یک برنامه ارائه می‌دهند و می‌توانند به تشخیص مشکلات قبل از تأثیر بر کاربران کمک کنند.

## تکنیک‌های فیلوور

تکنیک‌های فیلوور شامل تغییر به یک سیستم، سرور، شبکه یا جزء افزونه یا آماده به کار هنگامی که سیستم فعلی فعال شکست می‌خورد، است. این برای حفظ دسترسی و تداوم سرویس حیاتی است. در .NET، یک رویکرد رایج برای دستیابی به فیلوور استفاده از خوشبندی است.

## متعادل‌سازی بار

در .NET، متعادل‌سازی بار معمولاً می‌تواند در چندین لایه، از جمله DNS، سخت‌افزار و منطق برنامه مدیریت شود. متعادل‌سازی بار سطح نرم‌افزار می‌تواند با توزیع درخواست‌ها در میان یک استخراج سرورها یا سرویس‌ها بر اساس الگوریتم‌های مختلف مانند چرخشی، کمترین اتصالات، یا حتی طرح‌های انطباقی پیچیده‌تر که بار سرور یا زمان‌های پاسخ را در نظر می‌گیرند، انجام شود.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>



# تکنیک‌های سریالی‌سازی داده‌ها در C# 12 و .NET 8

در برنامه‌نویسی شبکه با C# 12 و .NET 8، سریالی‌سازی کارآمد داده‌ها یک نیاز اساسی است. این فرآیند تبدیل ساختارهای داده یا وضعیت اشیاء به فرمتی است که می‌تواند ذخیره یا منتقل شده و بعداً بازسازی شود.

# مفاهیم اصلی و اصطلاحات سریالیسازی داده‌ها

2

## فرمتر (Formatter)

فرمتر مؤلفه‌ای است که تعریف می‌کند چگونه یک شیء به فرمتی مانند XML یا JSON رمزگذاری و سپس به یک شیء رمزگشایی می‌شود. فرمترهای بومی مانند `XmlSerializer` و `JsonSerializer` در .NET ارائه می‌دهد.

1

## سریالیسازی و دیسریالیسازی

سریالیسازی فرآیند تبدیل وضعیت و ساختار یک شیء به فرمتی است که قابل ذخیره‌سازی در فایل، حافظه یا ارسال از طریق شبکه باشد. دیسریالیسازی فرآیند معکوس است که در آن جریان بایت یا فایل به یک شیء تبدیل می‌شود.

4

## قرارداد داده (Data Contract)

یک توافق رسمی که ساختار داده را برای سریالیسازی تعریف می‌کند و سازگاری و همخوانی را در سیستم‌های مختلف تضمین می‌کند. قراردادهای داده برای مدیریت نسخه‌بندی و تکامل طرح در سیستم‌های توزیع شده مفید هستند.

3

## گراف اشیاء (Object Graph)

این اصطلاح به اشیاء به هم پیوسته اشاره دارد؛ گراف با یک شیء ریشه شروع می‌شود و تمام اشیاء قابل دسترس از این ریشه را در بر می‌گیرد. سریالیسازی کل گراف را پردازش می‌کند، نه فقط اشیاء منفرد.

# مقدمه‌ای بر سریالی‌سازی داده‌ها در C# و .NET.

سریالی‌سازی داده‌ها در C# و .NET شامل تبدیل یک شیء یا ساختار داده به فرمتی است که به راحتی قابل ذخیره‌سازی، انتقال و بازسازی بعدی باشد. این فرآیند برای برنامه‌نویسی شبکه ضروری است، جایی که داده‌ها باید بین اجزا یا سیستم‌هایی که ممکن است معماری داخلی یکسانی نداشته باشند، منتقل شوند.

چندین مکانیسم سریالی‌سازی داخلی را ارائه می‌دهد که از فرمت‌های .NET مختلفی پشتیبانی می‌کنند که نیازهای خاصی را برآورده می‌کنند، مانند فرمت‌های XML و JSON.



# سربالی‌سازی JSON در C# و .NET

سربالی‌سازی JSON به‌ویژه در سرویس‌های وب و API‌ها محبوب است. به دلیل خوانایی و سبک بودن آن، برای انتقال شبکه بسیار مهم است. C# و .NET با فضای نام System.Text.Json، سربالی‌سازی JSON را ساده می‌کنند.

```
using System.Text.Json;
public class Person{
    public string? Name { get; set; }
    public int Age { get; set; }
}
public class Program{
    public static void Main()
    {
        var person = new Person { Name = "John Doe", Age = 30 };
        var jsonString =
JsonSerializer.Serialize(person);
        Console.WriteLine(jsonString);
    }
}
```

در این مثال، یک شیء Person ایجاد شده و با استفاده از JsonSerializer به رشته JSON تبدیل می‌شود. این روش ساده و کارآمد برای تبدیل اشیاء به فرمت JSON است که می‌تواند به راحتی منتقل یا ذخیره شود.  
<https://www.linkedin.com/in/mohamadhosseini/>  
<https://github.com/MohamadHoseinRoohiAmini>

# سریالی‌سازی XML در C# و .NET

سریالی‌سازی XML می‌تواند به‌ویژه هنگام کار با سیستم‌های قدیمی که به فرمات‌های داده XML نیاز دارند یا هنگامی که خوانایی انسانی و اعتبارسنجی سند مهم است، مفید باشد. فضای نام ابزارهایی برای تبدیل اشیاء به XML و بالعکس ارائه می‌دهد.

```
using System.Xml.Serialization; public class Person{    public string? Name { get; set; }    public int Age { get; set; }} public class Program{    public static void Main() {        var person = new Person { Name = "Jane Doe", Age = 28 };        var serializer = new XmlSerializer(typeof(Person));        using var stringWriter = new StringWriter();        serializer.Serialize(stringWriter, person);        Console.WriteLine(stringWriter.ToString());    }}
```

<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

در این مثال، یک شیء Person با استفاده از XmlSerializer به XML تبدیل می‌شود. این روش برای سیستم‌هایی که به فرمات XML نیاز دارند یا برای اسناد ساختاریافته مناسب است.

# انتخاب روش سریالیسازی مناسب

## سریالیسازی XML

برای تعاملات مبتنی بر سند مانند سرویس‌های وب SOAP یا پیکربندی‌ها. امکان کنترل دقیق بر نحوه تبدیل اشیاء به XML و بالعکس را فراهم می‌کند.

## عملکرد

سرعت، اندازه، سازگاری و سهولت استفاده از عوامل مهم در انتخاب روش سریالیسازی هستند. هر روش مزايا و معایب خود را دارد که باید با توجه به نیازهای برنامه در نظر گرفته شوند.

## سریالیسازی JSON

برای API‌های وب و سرویس‌هایی که قابلیت همکاری بین سیستمی مهم است JSON. هم خوانا برای انسان و هم به طور گسترده در فناوری‌های مختلف پشتیبانی می‌شود.

## سازگاری

انتخاب روش سریالیسازی مناسب به درک نیازهای برنامه و مبادلات مرتبط با هر نوع سریالیسازی بستگی دارد، که عملکرد و سازگاری بهینه را تضمین می‌کند.



# عوامل تأثیرگذار بر انتخاب روش سریالیسازی

## قابلیت همکاری بین سیستمی

هنگامی که برنامه نیاز به ارتباط با سیستم‌های دیگری دارد که ممکن است از .NET استفاده نکند، قابلیت همکاری بین سیستمی بسیار مهم است. سریالیسازی JSON و XML در چنین مواردی مناسب‌تر هستند زیرا این فرمتهای به راحتی در پلتفرم‌ها و زبان‌های مختلف قابل استفاده هستند.

در اینجا نحوه های وب استفاده می‌شود API به دلیل ماهیت سبک و خوانایی آن، به طور گسترده در JSON آمده است JSON پیاده‌سازی سریالیسازی:

```
using System.Text.Json; public class InteroperableData{    public int Id {  
    get; set; }    public string? Detail { get; set; }} public class  
InteropSerialization{    public static void Main()    {        var data =  
new InteroperableData { Id = 1, Detail = "Accessible" };        var  
jsonData = JsonSerializer.Serialize(data);  
Console.WriteLine(jsonData);    }}
```

## ملاحظات امنیتی

ملاحظات امنیتی نیز نقش مهمی دارند، به ویژه هنگامی که داده‌های حساس درگیر هستند. انتخاب فرمت سریالیسازی که برنامه را در معرض آسیب‌پذیری‌های امنیتی مانند آنچه در برخی تجزیه‌کننده‌های XML یافت می‌شود (مانند حملات XML External Entity) قرار ندهد، مهم است. علاوه بر این، روش سریالیسازی باید از مکانیسم‌هایی برای مدیریت امن داده‌ها، رمزگذاری یا مبهم‌سازی در صورت لزوم پشتیبانی کند.

<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

# دستورالعمل‌های عملی و توصیه‌ها

## در نظر گرفتن پیامدهای امنیتی

به آسیب‌پذیری‌های امنیتی مرتبط با سریالی‌سازی توجه کنید. داده‌های حساس را فقط با اقدامات امنیتی مناسب مانند رمزگذاری یا توکن‌سازی سریالی کنید. هنگام استفاده از سریالی‌سازی XML، در برابر حملات External Entity (XXE) با خاموش کردن پردازش DTD و اعتبارسنجی طرح در تجزیه‌کننده‌های XML محافظت کنید.

## استفاده از فرمت سریالی‌سازی مناسب برای سناریوی مناسب

همیشه فرمت سریالی‌سازی را بر اساس نیازهای خاص برنامه خود انتخاب کنید. به عنوان مثال، اگر برنامه شما با سیستم‌های خارجی یا مشتریان وب ارتباط برقرار می‌کند، JSON اغلب به دلیل پشتیبانی گسترده و خوانایی آن ترجیح داده می‌شود.

## پیاده‌سازی دیسریالی‌سازی قوی

دیسریالی‌سازی باید با دقت انجام شود تا از فساد داده‌ها و خطرات امنیتی جلوگیری شود. همیشه داده‌های ورودی را اعتبارسنجی کنید و استثناهای را به طور مناسب مدیریت کنید تا از خرابی برنامه جلوگیری شود. استفاده از قراردادهای داده و نسخه‌بندی را برای مدیریت تغییرات در ساختارهای داده در طول زمان در نظر بگیرید، تا سازگاری رو به عقب تضمین شود.

## روشن کردن مفهوم بارگذاری تنبل در سریالی‌سازی

برای بهبود عملکرد سریالی‌سازی، کاهش اندازه داده‌های در حال سریالی‌سازی را در نظر بگیرید. این کار می‌تواند با حذف فیلدهای اضافی یا نامربوط از سریالی‌سازی انجام شود. علاوه بر این، از ویژگی‌هایی مانند بارگذاری تنبل، تکنیکی که بارگذاری داده‌های غیرضروری را تا زمانی که واقعاً مورد نیاز باشد به تعویق می‌اندازد، برای مجموعه داده‌های بزرگ استفاده کنید.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# مثال عملی: حذف ویژگی‌ها از سریالیسازی JSON

```
using System.Text.Json;using System.Text.Json.Serialization;void Main(){    var config = new Configuration { Username = "Admin", Password = "12345", RefreshInterval = 300 };    string jsonString = JsonSerializer.Serialize(config);    Console.WriteLine(jsonString);}public class Configuration{    public string? Username { get; set; }    [JsonIgnore]    public string? Password { get; set; } // از سریالیسازی حذف شده است    public int RefreshInterval { get; set; }}
```

در این مثال، ویژگی `Password` با استفاده از `[JsonIgnore]` از سریالیسازی حذف شده است. این یک روش مهم برای محافظت از داده‌های حساس مانند رمزهای عبور است که نباید سریالی شوند یا در خروجی JSON ظاهر شوند.

# مثال عملی: دیسریالیسازی قوی

```
using System.Text.Json;public class User{    public string Name { get; init; }    public int Age { get; init; }}public class Program{    public static void Main()    {        // و سریالیسازی آن به User ایجاد یک نمونه از کلاس JSON        var user = new User { Name = "Alice", Age = 28 };        string jsonString = JsonSerializer.Serialize(user);        Console.WriteLine($"Serialized JSON: {jsonString}");        // JSON دیسریالیسازی رشته User        var deserializedUser = DeserializeUser(jsonString);        if (deserializedUser != null)        {            Console.WriteLine($"Deserialized User: Name = {deserializedUser.Name}, Age = {deserializedUser.Age}");        }    }    private static User? DeserializeUser(string jsonString)    {        try        {            return JsonSerializer.Deserialize(jsonString);        }        catch (JsonException ex)        {            Console.WriteLine($"Deserialization failed: {ex.Message}");            return null; // مدیریت خطا به طور مناسب        }    }}
```

ht

<https://github.com/MohamadHoseinRoohiAmini>

این مثال نشان می‌دهد که چگونه می‌توان دیسریالیسازی را با مدیریت خطا انجام داد. متدهای `DeserializeUser` و `SerializeUser` به یک شیء `User` دیسریالیسازی می‌کند و در صورت بروز خطا، استثنای گرفته و پیام خطا را افایش می‌دهد. این روش برای جلوگیری از خرابی برنامه در صورت دریافت داده‌های نامعتبر مهم است.

# کارایی در ساختارهای داده و طراحی

کارایی ساختار داده و طراحی برای بهینه‌سازی فرآیندهای سریالی‌سازی و دیسریالی‌سازی در برنامه‌های شبکه با استفاده از C# و .NET بسیار مهم است. ساختارهای داده طراحی شده به خوبی، مقدار داده‌های منتقل شده از طریق شبکه را کاهش می‌دهند و سرعت سریالی‌سازی و دیسریالی‌سازی را افزایش می‌دهند، که برای حفظ عملکرد بالا در سیستم‌های توزیع شده حیاتی است.

ساده و مسطح نگه داشتن ساختارهای داده برای دستیابی به سریالی‌سازی کارآمد بسیار مهم است. گراف‌های اشیاء پیچیده یا عمیقاً تو در تو می‌توانند فرآیند سریالی‌سازی را به طور قابل توجهی کند کرده و اندازه داده‌های سریالی شده را افزایش دهنند.

```
public class UserActivity{    public DateTime Timestamp {  
    get; set; }    public string? ActivityType { get; set; }  
    public string? Username { get; set; }}public static void  
SerializeUserActivity(UserActivity activity){    string  
jsonString =  
System.Text.Json.JsonSerializer.Serialize(activity);  
Console.WriteLine(jsonString);}
```

در این مثال، کلاس `UserActivity` یک ساختار داده ساده و مسطح است که برای سریالی‌سازی بهینه‌سازی شده است. این کلاس فقط شامل سه ویژگی است که همگی انواع داده ساده هستند، که سریالی‌سازی و دیسریالی‌سازی را سریع‌تر و کارآمدتر می‌کند.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# استفاده از ویژگی‌های پیشرفته سریالی‌سازی



## سریالی‌سازی ناهمگام

سریالی‌سازی ناهمگام در C# و .NET از مدل برنامه‌نویسی ناهمگام برای انجام وظایف سریالی‌سازی بدون مسدود کردن رشته اصلی برنامه استفاده می‌کند. این تکنیک در محیط‌های با بار بالا با پاسخگویی بحرانی، مانند برنامه‌های وب یا سیستم‌های پردازش داده در زمان واقعی، ارزشمند است.

4

## فراخوانی‌های سریالی‌سازی

همچنین از فراخوانی‌های سریالی‌سازی پشتیبانی می‌کند، که متدهایی هستند C# که به طور خودکار در طول فرآیند سریالی‌سازی یا دیسریالی‌سازی فراخوانی ، OnSerialized، OnDeserializing، OnDeserializing، OnSerialized) این فراخوانی‌ها می‌شوند به توسعه‌دهندگان اجازه می‌دهند که را در مراحل مختلف فرآیند (OnDeserialized سریالی‌سازی اجرا کنند.

## استراتژی‌های کش کردن

استراتژی‌های کش کردن برای سریالی‌سازی در C# و .NET می‌توانند سربار سریالی‌سازی و دیسریالی‌سازی مکرر همان اشیاء را به طور قابل توجهی کاهش دهند. با ذخیره اشیاء سریالی شده در حافظه، برنامه‌ها می‌توانند به سرعت این داده‌ها را بازیابی و مجدد استفاده کنند بدون فرآیندهای سریالی‌سازی اضافی.



## منطق سریالی‌سازی سفارشی

مکانیسم‌هایی را برای پیاده‌سازی منطق سریالی‌سازی سفارشی با استفاده از C# این امکان کنترل دقیق بر نحوه ارائه می‌دهد Serializable ارابط‌هایی مانند سریالی‌سازی و دیسریالی‌سازی اشیاء را فراهم می‌کند، که سناریوهای پیچیده‌ای مانند حفظ مراجع اشیاء، مدیریت نسخه‌بندی یا سریالی‌سازی فیلدهای خصوصی را پوشش می‌دهد.

# مثال عملی: استراتژی‌های کش کردن

```
using System.Text.Json;
public class User{
    public string? Name { get; set; }
    public int Age { get; set; }
}
public class SerializationCache
{
    private static readonly Dictionary<int, string> Cache = new Dictionary<int, string>();
    public static string? SerializeUser(User user, int userId)
    {
        if (Cache.TryGetValue(userId, out string? cachedJson))
        {
            Console.WriteLine("Cache hit");
            return cachedJson;
        }
        string? jsonString = JsonSerializer.Serialize(user);
        Cache[userId] = jsonString;
        Console.WriteLine("Cache miss - adding to cache");
        return jsonString;
    }
    public static void Main()
    {
        var user = new User { Name = "Alice", Age = 28 };
        string? json1 = SerializeUser(user, 1);
        // Cache miss
        Console.WriteLine(json1);
        string? json2 = SerializeUser(user, 1);
        // Cache hit
        Console.WriteLine(json2);
    }
}
```

# تست عملکرد و نظارت

## تست عملکرد

تست عملکرد سریالی‌سازی شامل اندازهگیری زمان لازم برای سریالی‌سازی و دیسریالی‌سازی اشیاء و اندازه داده‌های سریالی شده است. این کار می‌تواند با استفاده از ابزارهای معیارسنجی یا به سادگی با نوشتن موارد آزمایشی سفارشی که این عملیات را در شرایط مختلف زمان‌بندی می‌کنند، انجام شود.

## نظارت

در محیط‌های تولید، نظارت مداوم بر عملکرد سریالی‌سازی به عنوان بخشی از نظارت بر سلامت کلی برنامه بسیار مهم است. این معمولاً شامل ثبت معیارهای کلیدی عملکرد در طول عملیات سریالی‌سازی و دیسریالی‌سازی و استفاده از ابزارهای نظارتی است که می‌توانند توسعه‌دهنده‌گان را از تغییرات ناگهانی یا کاهش عملکرد مطلع کنند.

```
using System.Text.Json;using System.Diagnostics;public class MonitorData{ public string? Information { get; set; }}public class MonitorSerialization{ public static void SerializeData(MonitorData data) { var stopwatch = new Stopwatch(); stopwatch.Start(); var jsonString = JsonSerializer.Serialize(data); stopwatch.Stop(); // Log("Serialization", stopwatch.ElapsedMilliseconds, jsonString.Length); } private static void Log(string operation, long time, long size) { // Console.WriteLine($"{operation}: Time = {time} ms, Size = {size} bytes"); }}
```

```
using System.Diagnostics;using System.Text.Json;public class SerializationPerformanceTest{ public static void Main() { var data = new TestData { Numbers = new int[10000] }; stopwatch.Start(); var stopwatch = new Stopwatch(); // var jsonData = JsonSerializer.Serialize(data); stopwatch.Stop(); Console.WriteLine($"Serialization time: {stopwatch.ElapsedMilliseconds} ms"); // JSON stopwatch.Restart(); var serializedData = JsonSerializer.Serialize(data); stopwatch.Stop(); Console.WriteLine($"Deserialization time: {stopwatch.ElapsedMilliseconds} ms"); }}public class TestData{ public int[] Numbers { get; set; }}
```

# خلاصه

## استراتژی‌های عملی

استراتژی‌های عملی برای افزایش کارایی سریالی‌سازی در پرداختن به بهینه‌سازی عملکرد نیز در این فصل مورد تأکید قرار گرفته است. این شامل بهینه‌سازی ساختارهای داده و طراحی، استفاده از ویژگی‌های پیشرفته سریالی‌سازی مانند سریالایزرهاي سفارشی و فراخوانی‌ها، و بهره‌گیری از ویژگی‌ها و ابزارهای قدرتمند سریالی‌سازی .NET است. این بخش غنی از نمونه کدها و نکاتی برای کاهش سربار، مدیریت مؤثر حافظه و به حداقل رساندن تأثیر بر منابع شبکه و سیستم است.

## انتخاب روش مناسب

این فصل همچنین به فرآیند تصمیم‌گیری برای انتخاب روش سریالی‌سازی مناسب پرداخته است. عواملی مانند عملکرد، اندازه داده‌ها، سازگاری و سهولت استفاده را در نظر می‌گیرد. این بخش بر اهمیت انتخاب فرمت سریالی‌سازی مناسب بر اساس نیازهای خاص برنامه تأکید می‌کند، خواه برای ارتباطات داخلی با عملکرد بالا، سرویس‌های قابل همکاری یا فرمتهای خوانا برای انسان مناسب برای پیکربندی و آزمایش باشد.

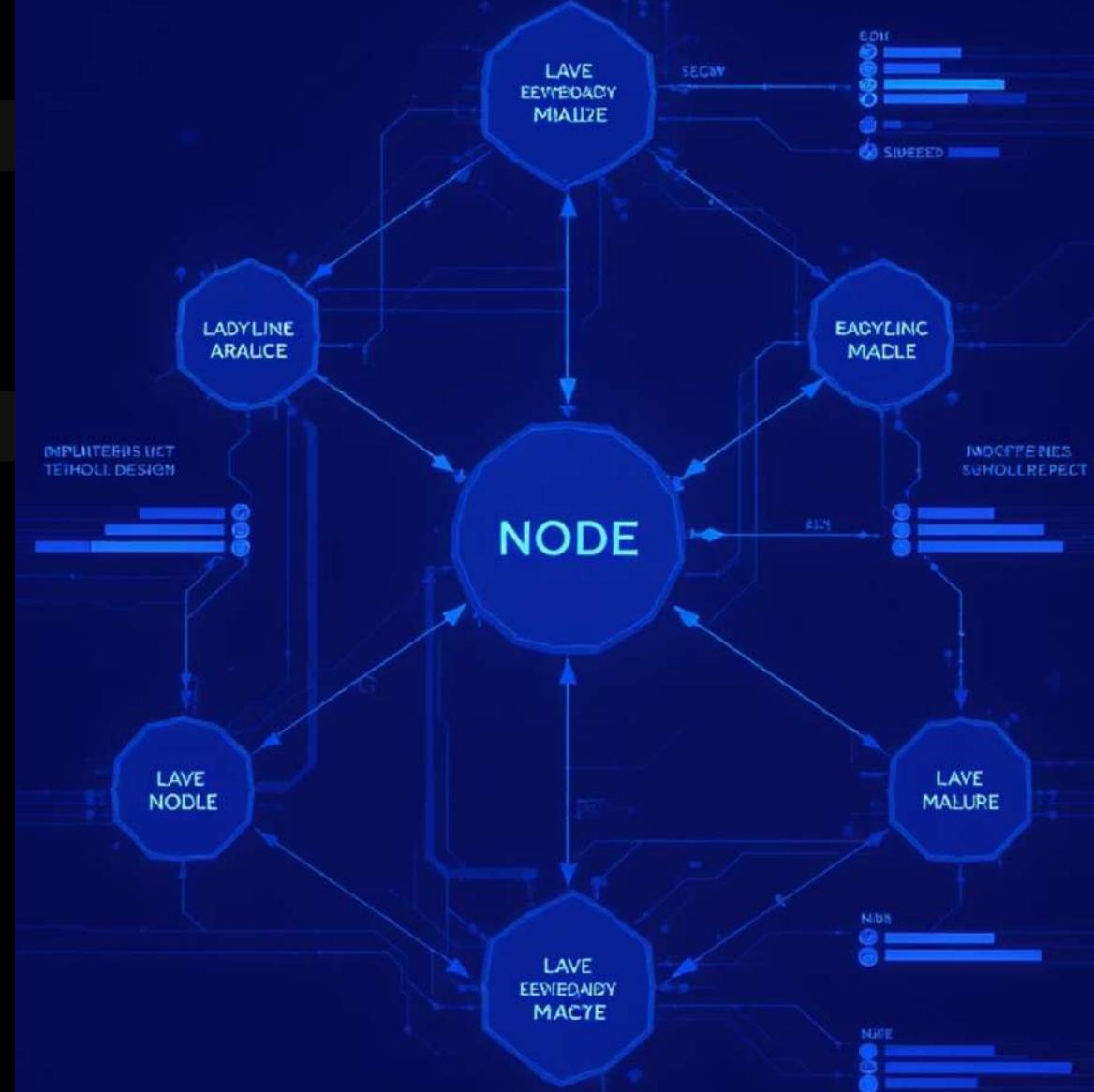
## مفاهیم اساسی

این فصل یک راهنمای جامع برای درک و پیاده‌سازی سریالی‌سازی در برنامه‌های نرم‌افزاری مدرن است. با معرفی مفاهیم اساسی سریالی‌سازی، از جمله مکانیسم‌های پایه ارائه شده توسط .NET. شروع می‌شود. این دانش پایه برای توسعه‌دهندگان جهت درک ابزارها و روش‌های مختلف برای تبدیل داده‌ها به فرمتی مناسب برای ذخیره‌سازی یا انتقال از طریق شبکه‌ها ضروری است.

# بهینه‌سازی عملکرد شبکه

راهنمای جامع برای توسعه‌دهندگان .NET و C# برای بهبود کارایی برنامه‌های شبکه

NETWORK PERFORMANCE  
SEFT FREST OPTIMIZATION



<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# مقدمه: اهمیت بهینهسازی عملکرد شبکه

## اهداف این ارائه

این ارائه به تکنیک‌های عملی بهینهسازی، از جمله استفاده هوشمندانه از چندنخی و موازی‌سازی وظایف، بهینهسازی پروتکل‌های شبکه و فشرده‌سازی استراتژیک داده‌ها می‌پردازد. ما مطالعات موردی و معیارهای عملکرد را برای نشان دادن تأثیر این بهینهسازی‌ها در سناریوهای دنیای واقعی بررسی خواهیم کرد.

## چالش‌های پیش رو

بهینهسازی عملکرد شبکه در برنامه‌های .NET کاری پیچیده است که شامل رویکردهای متنوعی می‌شود. استفاده بهینه از ویژگی‌های C#، مانند مدل‌های برنامه‌نویسی ناهمگام بهبودیافته و دسترسی به حافظه مبتنی بر `span`، می‌تواند زمان و منابع مورد نیاز برای ارتباطات شبکه را کاهش دهد.

## اهمیت بهینهسازی

بهینهسازی عملکرد شبکه برای توسعه برنامه‌های قوی و کارآمد در برنامه‌نویسی شبکه با استفاده از .NET و C# بسیار مهم است. این فصل به دنبال معرفی استراتژی‌های پیشرفته برای افزایش کارایی و پاسخگویی عملیات شبکه است.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# درک و تحلیل عملکرد شبکه در .NET

هنگامی که به سفر درک و تحلیل عملکرد شبکه در .NET می‌پردازیم، ایجاد پایه‌ای محکم از مفاهیم و معیارهای کلیدی که کارایی تعاملات شبکه را تعیین می‌کنند، بسیار مهم است. این بخش به دنبال تجهیز توسعه‌دهندگان با ابزارها و دانش لازم برای ارزیابی دقیق عملکرد برنامه‌های شبکه‌ای خود است.

با درک معیارهای اساسی مانند تأخیر، توان عملیاتی و از دست دادن بسته، توسعه‌دهندگان می‌توانند بینشی نسبت به جنبه‌های عملیاتی برنامه‌های خود به دست آورند و مناطقی را که ممکن است نیاز به بهینه‌سازی داشته باشند، شناسایی کنند.



<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

# ابزارها و تکنیک‌های تحلیل عملکرد



## نظارت مستمر

با تجزیه و تحلیل سیستماتیک این معیارها، توسعه‌دهندگان می‌توانند انتقال داده را بهینه کنند، سریار شبکه را کاهش دهند و عملکرد برنامه را بهبود بخشدند. ترکیب این تکنیک‌ها با نظارت دقیق و آزمایش مستمر عملکرد اطمینان می‌دهد که برنامه‌های شبکه کارآمد و مقیاس‌پذیر باقی می‌مانند.

## ابزارهای پروفایل و تشخیص

ابزارهایی مانند Visual Studio Event Tracing و Performance Profiler for Windows (ETW) می‌توانند می‌توانند معیارهای دقیقی در مورد فعالیت شبکه، استفاده از CPU و تخصیص حافظه ارائه می‌دهند. پروفایل‌گیری ابزاری قدرتمند است که به شناسایی کد ناکارآمد، فراخوانی‌های شبکه بیش از حد و سایر مشکلات عملکردی کمک می‌کند.

## برنامه‌نویسی ناهمگام

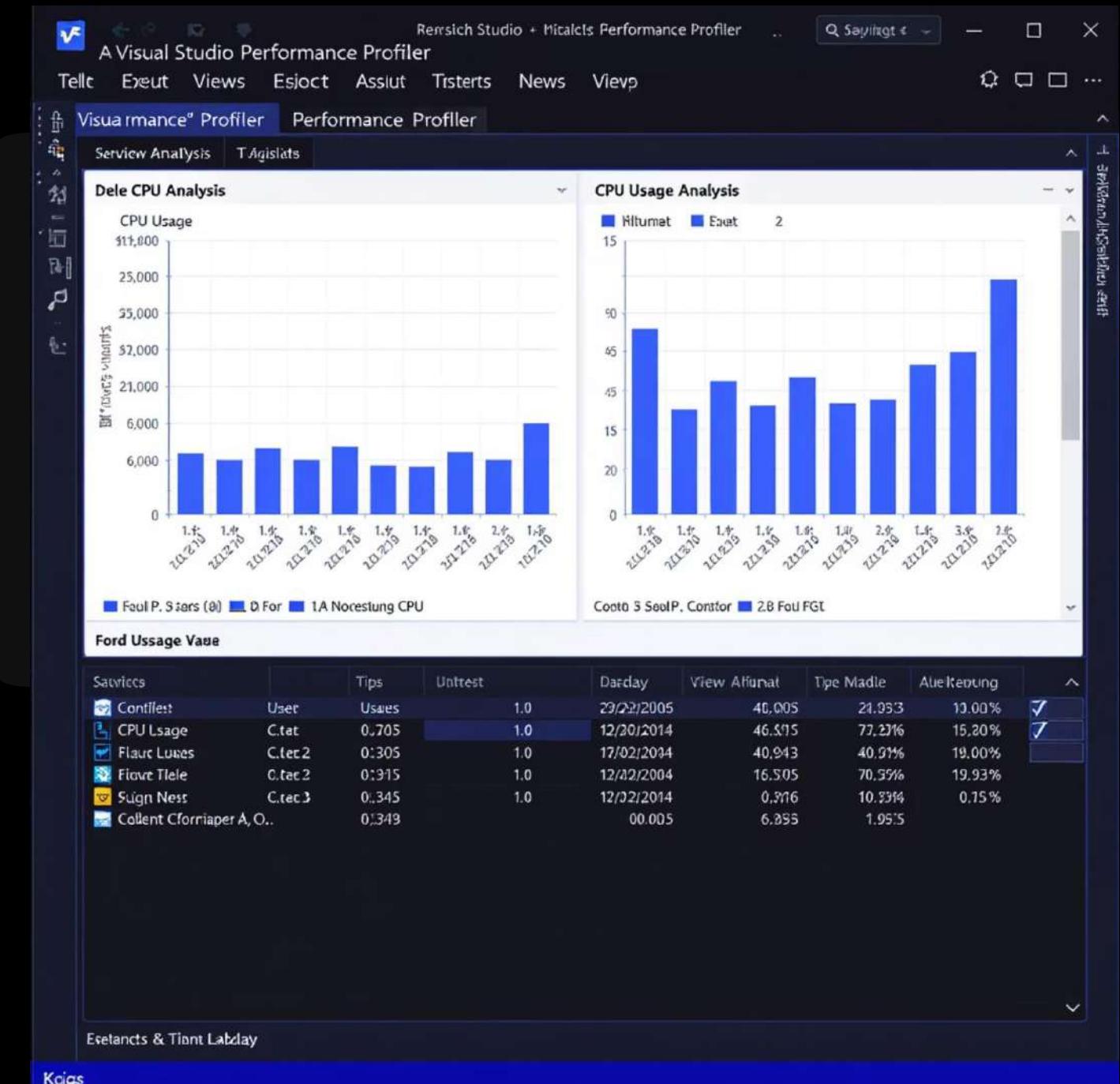
برنامه‌نویسی ناهمگام، که توسط کلیدواژه‌های `async` و `await` تسهیل می‌شود، به جلوگیری از فراخوانی‌های مسدودکننده شبکه کمک می‌کند و به برنامه‌ها اجازه می‌دهد چندین عملیات شبکه را به طور همزمان مدیریت کنند. این رویکرد تأخیر را کاهش می‌دهد و پاسخگویی کلی برنامه‌های شبکه را بهبود می‌بخشد.

# پروفایلر عملکرد Visual Studio

بهینه‌سازی عملکرد شبکه برای اطمینان از برنامه‌های پاسخگو و کارآمد بسیار مهم است. یکی از قدرتمندترین ابزارهای در دسترس توسعه‌دهندگان .NET، پروفایلر عملکرد Visual Studio است. این بخش راهنمای گام به گام استفاده از پروفایلر عملکرد Visual Studio برای شناسایی و رفع گلگاه‌های عملکرد در برنامه‌های شبکه شما را ارائه می‌دهد.

## گام ۱: راهاندازی پروفایلر

برای شروع پروفایل برنامه خود، پروژه خود را در Visual Studio باز کنید. به Debug > Performance Profiler بروید. لیستی از ابزارهای موجود را خواهید دید. ابزار CPU Usage را برای نظارت بر میزان زمان CPU صرف شده در قسمت‌های مختلف برنامه خود انتخاب کنید.



گام ۲: اجرای پروفایلر

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# تحلیل نتایج پروفایلر Visual Studio

## گام ۵: بهینهسازی کد

پس از شناسایی گلوگاهها، زمان آن است که تغییرات کد لازم را برای بهینهسازی عملکرد انجام دهید. در اینجا تخصص و درک شما از کد به کار می‌آید. ممکن است بهینهسازی سریالی‌سازی داده‌ها، کاهش تعداد فراخوانی‌های شبکه یا پیاده‌سازی الگوریتم‌های کارآمدتر را در نظر بگیرید.

## گام ۴: شناسایی گلوگاهها

گزارش را بررسی کنید تا متدهایی را که در طول عملیات شبکه مقدار قابل توجهی از زمان CPU را مصرف می‌کنند، شناسایی کنید. به دنبال هرگونه افزایش غیرمنتظره یا مناطقی باشید که در آن استفاده از CPU به طور نامتناسب بالا است. بهینهسازی‌ها در این نقاط داغ بیشترین تأثیر را خواهند داشت.

## گام ۳: تحلیل نتایج

پس از اتمام جلسه پروفایل‌گیری، برای پایان دادن به جلسه روی Stop Collection کلیک کنید. Visual Studio داده‌های جمع‌آوری شده را پردازش کرده و گزارش دقیقی نمایش می‌دهد. گزارش CPU Usage، به عنوان مثال، به شما نشان می‌دهد کدام متد‌ها بیشترین زمان CPU را مصرف می‌کنند، که به شما امکان می‌دهد گلوگاه‌های احتمالی در کد شبکه خود را شناسایی کنید.

```
public void FetchData(){ var client = new HttpClient(); var response = client.GetStringAsync("https://example.com").Result;
ProcessData(response);}//public async Task FetchDataAsync(){ var client = new HttpClient(); var response = await
client.GetStringAsync("https://example.com"); ProcessData(response);}
```

# JetBrains dotTrace

این بخش .ارائه می‌دهد .ابزار پروفایل قدرتمندی است که تحلیل عملکرد عمیقی برای برنامه‌های .NET برای شناسایی و حل گلوگاه‌های عملکرد در برنامه‌های شبکه را نشان می‌دهد، که به شما امکان dotTrace نحوه استفاده از می‌دهد که خود را به طور مؤثر بینه کنید.

## گام ۱: راهاندازی dotTrace

ابتدا اطمینان حاصل کنید که JetBrains dotTrace نصب شده است .پروژه خود را در Visual Studio باز کنید و Run | JetBrains Rider نوار ابزار را باز کنید .برای شروع پروفایل‌گیری، اگر از Rider استفاده می‌کنید، روی | Run | Attach to Process کلیک کنید، یا Run | Attach to Process | Profile Startup Project را برای پروفایل یک برنامه در حال اجرا انتخاب کنید.



گام ۲: اجرای پروفایل

<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

# تحلیل نتایج dotTrace

## گام ۵: بهینهسازی کد

تغییرات کد لازم را برای بهینهسازی گلوگاه‌های شناسایی شده انجام دهید. در مثال زیر، تغییر از `JsonConvert.DeserializeObject` به `JsonSerializer.Deserialize` می‌تواند عملکرد تجزیه را به دلیل کارایی بهتر و سریار کمتر بهبود بخشد. پس از ایجاد تغییرات، جلسه پروفایل‌گیری را مجددًا اجرا کنید تا بهبودها را تأیید کنید.

## گام ۴: شناسایی گلوگاه‌ها

در درخت فراخوانی، به دنبال متدهایی با زمان اجرای بالا یا فراخوانی‌های مکرر باشید. اینها گلوگاه‌های بالقوه هستند. به عنوان مثال، اگر متوجه شدید که متدى مسئول تجزیه داده‌های JSON از پاسخ‌های شبکه زمان قابل توجهی می‌گیرد، نشان‌دهنده نیاز به بهینهسازی است.

## گام ۳: تحلیل نتایج

پس از اتمام جلسه پروفایل‌گیری، پروفایل را متوقف کنید تا داده‌های جمع‌آوری شده را مشاهده کنید. تصویری از عملکرد برنامه ارائه می‌دهد و متدهایی که بیشترین زمان را مصرف می‌کنند، برجسته می‌کند. نمای درخت فراخوانی به ویژه برای درک چگونگی انتشار فراخوانی‌های متدد در برنامه شما و مکان صرف زمان مفید است.

```
// قبل از بهینه‌سازی
public void ParseJsonData(string jsonData){ var jsonObject = JsonConvert.DeserializeObject<JSON>(jsonData);
ProcessData(jsonObject);}// پس از تجزیه استفاده از تجزیه بهینه‌شده JSON
public void ParseJsonDataOptimized(string jsonData){ var jsonObject =
JsonSerializer.Deserialize<JSON>(jsonData); ProcessData(jsonObject);}
```

http://

<https://github.com/MohamadHoseinRoohiAmini>

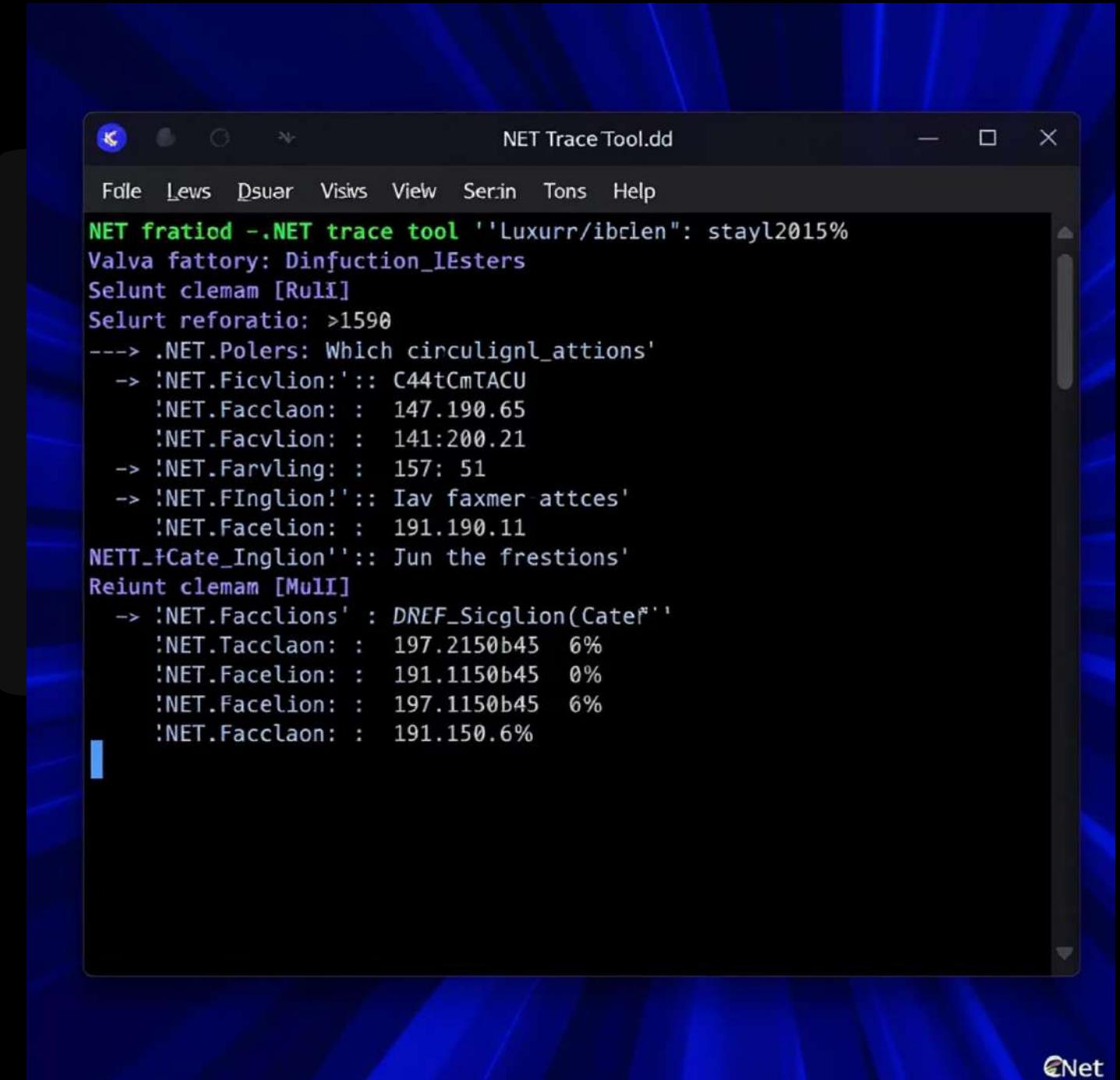
# NET. Trace

ابزار.NET Trace یک ابزار خط فرمان قدرتمند است که به توسعه‌دهندگان کمک می‌کند داده‌های عملکرد را برای برنامه‌های.NET ضبط و تحلیل کنند. این بخش نحوه استفاده از ابزار.NET Trace برای شناسایی گلوگاه‌های عملکرد در برنامه‌های شبکه را نشان می‌دهد و رویکردنی عملی برای بهینه‌سازی کد شما ارائه می‌دهد.

## گام ۱: راه‌اندازی

ابتدا اطمینان حاصل کنید که روی سیستم شما نصب شده است. ابزار.NET Trace در.NET SDK نجandه شده است. برای تأیید نصب، یک خط فرمان باز کنید و دستور زیر را اجرا کنید:

```
dotnet --version
```



The screenshot shows a terminal window titled "NET Trace Tool.dd" with the following command and its output:

```
NET fratioc -.NET trace tool ''Luxurr/ibclen": stayl2015%
Valva fattory: Dinfuction_lesters
Selunt clemam [Ruli]
Selurt reforatio: >1590
---> .NET.Polers: Which circulignl_attions'
-> :NET.Ficvlion:':: C44tCmTACU
:NET.Facclaon: : 147.190.65
:NET.Facvlion: : 141:200.21
-> :NET.Farvling: : 157: 51
-> :NET.FInglion'': Iav faxmer attces'
:NET.Facelion: : 191.190.11
NETT_FCatte_Inglion'': Jun the frestions'
Reiunt clemam [MuI]
-> :NET.Facclions' : DREF_Sicglion(Cate'')
:NET.Tacclaon: : 197.2150b45 6%
:NET.Facelion: : 191.1150b45 0%
:NET.Facelion: : 197.1150b45 6%
:NET.Facclaon: : 191.150.6%
```

# کد نمونه برای NET. Trace

```
public class NetworkOperation{    public async Task FetchAndProcessDataAsync()    {        var httpClient = new HttpClient();        var response = await  
httpClient.GetStringAsync("https://api.example.com/data");        ProcessData(response);    }    private void ProcessData(string data)    {        //        System.Threading.Thread.Sleep(200);        Console.WriteLine("Data processed.");    }    public static async Task Main(string[] args)    {        var networkOperation = new  
NetworkOperation();        await networkOperation.FetchAndProcessDataAsync();    }}
```

## گام ۵: بهینه‌سازی کد

پس از شناسایی گلوگاه، کد را برای بهبود عملکرد بهینه کنید. به عنوان مثال، می‌توانید همگام را با یک تأخیر ناهمگام جایگزین کنید تا از مسدود کردن رشته Thread.Sleep استفاده کنید تا داده‌های جمع‌آوری شده را به فرمتی که خواندن آن آسان‌تر است، مانند فرمت speedscope تبدیل کنید. فایل رديابي تبدیل شده را با استفاده از ابزاری مانند Speedscope باز کنید، گه نمایش بصری از داده‌های عملکرد ارائه می‌دهد و شناسایی گلوگاهها را آسان‌تر می‌کند.

<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

## گام ۴: تحلیل ردیابی

برای تحلیل ردیابی، از ابزار dotnet trace استفاده کنید تا داده‌های جمع‌آوری شده را به فرمتی که خواندن آن آسان‌تر است، مانند فرمت speedscope تبدیل کنید. فایل رديابي تبدیل شده را با استفاده از ابزاری مانند Speedscope باز کنید، گه نمایش بصری از داده‌های عملکرد ارائه می‌دهد و شناسایی گلوگاهها را آسان‌تر می‌کند.

## گام ۳: توقف و ذخیره ردیابی

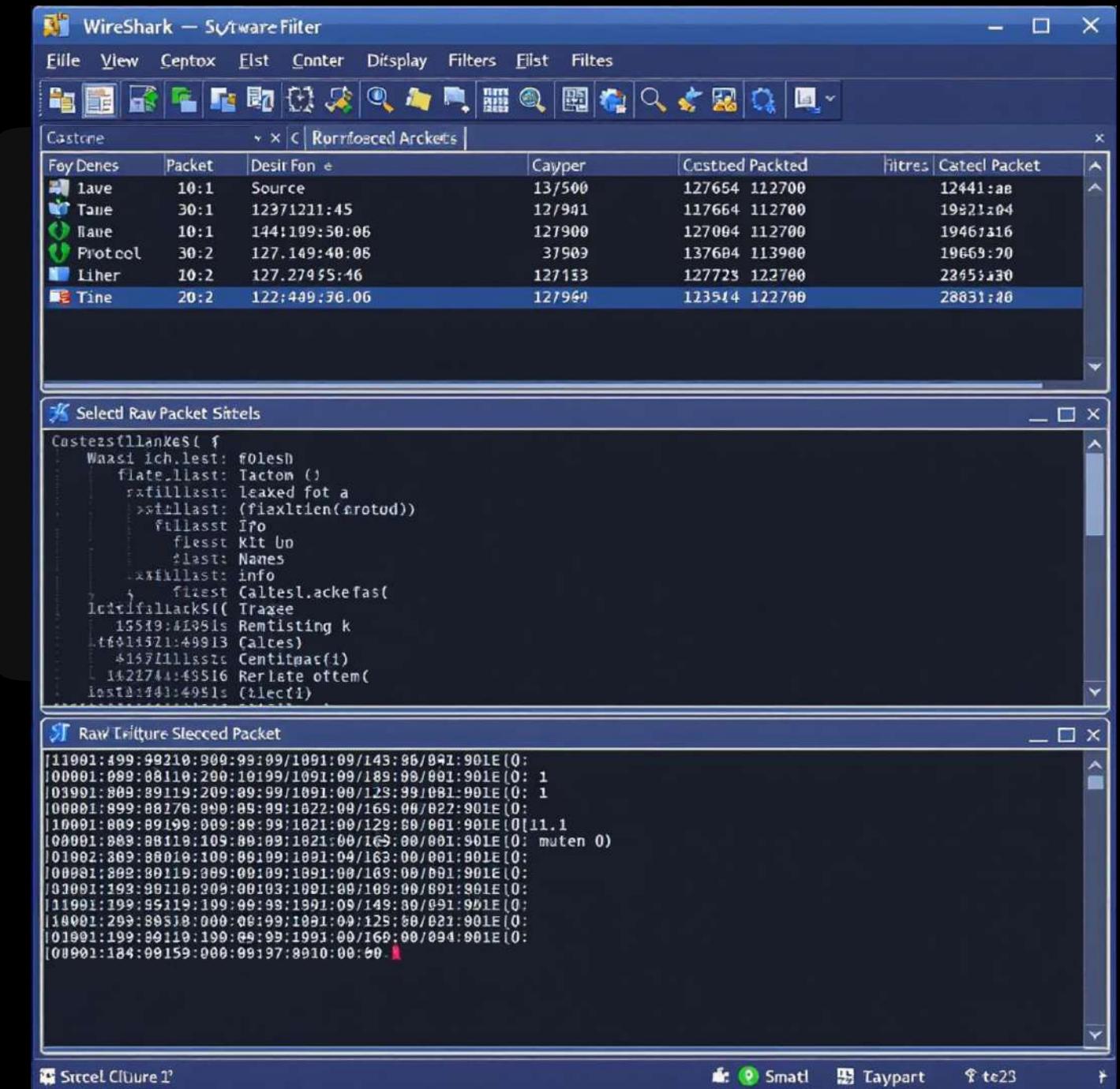
پس از جمع‌آوری داده‌های کافی، با فشار دادن Ctrl+C در خط فرمان، ردیابی را متوقف کنید. ابزار فایل ردیابی (مثلًا trace.netracer) را در دایرکتوری فعلی ذخیره می‌کند. این فایل حاوی داده‌های عملکرد دقیقی است که می‌توانید تحلیل کنید.

# WireShark

یک تحلیلگر پروتکل شبکه است که به طور گسترده استفاده می‌شود و بینش دقیقی از ترافیک شبکه WireShark. این ابزار برای تشخیص مشکلات شبکه و بهینه‌سازی عملکرد برنامه‌های شبکه بسیار ارزشمند است. ارائه می‌دهد برای شناسایی و حل گلوگاه‌های عملکرد در برنامه‌های شبکه شما را نشان WireShark این بخش نحوه استفاده از می‌دهد.

## گام ۱: راهاندازی

ابتدا، WireShark را از وبسایت رسمی (<https://www.wireshark.org/>) دانلود و نصب کنید. پس از نصب، WireShark را راهاندازی کنید و لیستی از رابطهای شبکه موجود به شما نمایش داده می‌شود. رابط برنامه خود را برای ارتباط شبکه (مثلًا Wi-Fi یا Ethernet) انتخاب کنید.



گام ۲: ضبط ترافیک شبکه

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# تحلیل ترافیک با WireShark

## گام ۵: شناسایی و حل گلوگاهها

بر اساس تحلیل، علل اصلی مشکلات عملکرد را شناسایی کنید. به عنوان مثال، اگر تأخیر در زمان‌های پاسخ سرور مشاهده می‌کنید، بهینه‌سازی کد سمت سرور را در نظر بگیرید. اگر از دست دادن بسته یا ارسال مجدد قابل توجهی وجود دارد، پایداری شبکه یا مشکلات پهنای باند را بررسی کنید.

## گام ۴: تحلیل ترافیک

بسته‌های فیلتر شده را برای شناسایی مشکلات عملکرد بررسی کنید. به دنبال تأخیر بالا در جفت‌های درخواست-پاسخ، از دست دادن بسته یا ارسال مجدد باشید. به عنوان مثال، زمان‌های پاسخ بالا برای درخواست‌های HTTP می‌توانند نشان‌دهنده گلوگاه عملکرد در سرور یا مسیر شبکه باشد.

## گام ۳: فیلتر کردن داده‌های ضبط شده

مقدار زیادی داده ضبط می‌کند، WireShark بنابراین فیلترها برای محدود کردن بسته‌های به عنوان مثال، اگر برنامه مرتبط ضروری هستند شما با یک سرور خاص ارتباط برقرار می‌کند، سرور فیلتر کنید IP‌آمی‌توانید بسته‌ها را با آدرس

```
ip.addr == 192.168.1.1
```

یا، اگر می‌خواهید ترافیک TCP را فیلتر کنید، می‌توانید از این استفاده کنید:

```
tcp
```

# معیارهای عملکرد شبکه

ایجاد معیارهای مؤثر عملکرد شبکه برای نظارت و بهینه‌سازی برنامه‌های شبکه ضروری است. این معیارها بینش ارزشمندی در مورد رفتار و کارایی عملیات شبکه ارائه می‌دهند و به توسعه‌دهندگان امکان می‌دهند گلوگاه‌های عملکرد را شناسایی و برطرف کنند.

3

## از دست دادن بسته (Packet Loss)

از دست دادن بسته زمانی رخ می‌دهد که بسته‌های داده نتوانند به مقصد خود برسند. این می‌تواند تأثیر شدیدی بر قابلیت اطمینان و کیفیت ارتباطات شبکه داشته باشد. ردیابی از دست دادن بسته به تشخیص مشکلات پایداری شبکه و اطمینان از یکپارچگی داده کمک می‌کند.

2

## توان عملیاتی (Throughput)

توان عملیاتی داده‌های منتقل شده از طریق شبکه در یک دوره زمانی معین را نشان می‌دهد. این نشان‌دهنده ظرفیت شبکه برای مدیریت کارآمد ترافیک داده است.

نظارت بر توان عملیاتی به شناسایی گلوگاه‌ها و بهینه‌سازی انتقال داده کمک می‌کند.

1

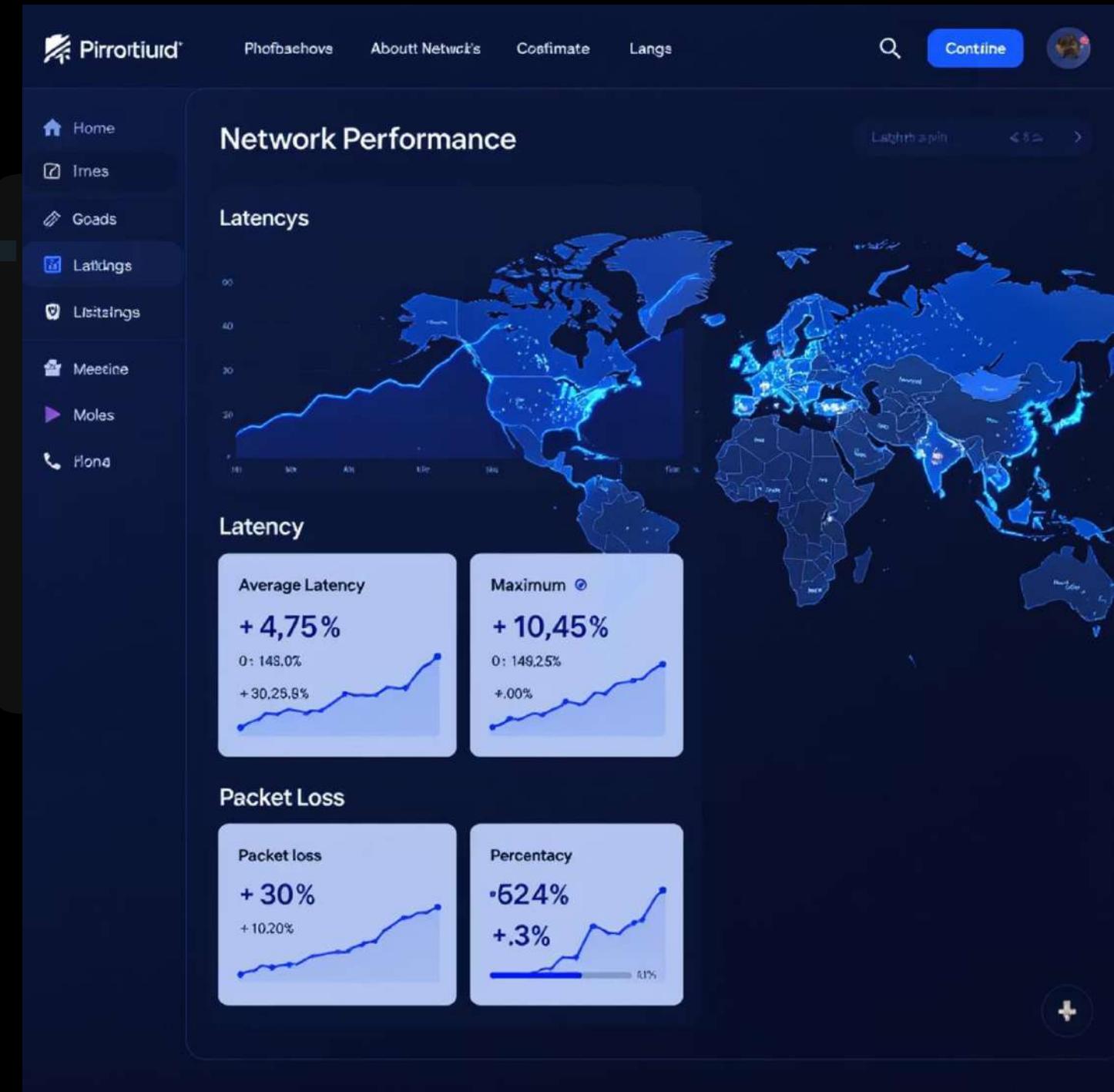
## تأخیر (Latency)

تأخیر زمانی را اندازه‌گیری می‌کند که طول می‌کشد تا داده از منبع به مقصد برسد. این یک شاخص مهم از پاسخگویی برنامه‌های شبکه است. تأخیر بالا می‌تواند تأثیر منفی بر تجربه کاربر داشته باشد، به ویژه در برنامه‌های زمان واقعی.

# پیادهسازی معیارهای عملکرد شبکه

نظرات بر معیارهای کلیدی عملکرد شبکه برای اطمینان از کارایی و قابلیت اطمینان برنامه‌های شبکه در #C و .NET بسیار مهم است. در این بخش، کد نمونه‌ای برای ردیابی تأخیر با استفاده از شمارنده‌های عملکرد سفارشی ارائه شده است.

```
using System.Diagnostics; using static System.Diagnostics.PerformanceCounterCategory; public class LatencyTracker { private readonly PerformanceCounter latencyCounter; public LatencyTracker() { if (!Exists("NetworkCategory")) { CounterCreationDataCollection counters = new(); counters.Capacity = 0; CounterCreationData latencyData = new() { CounterType = PerformanceCounterType.NumberOfItems32, CounterName = "Latency" }; latencyData.CounterHelp = "Network latency in milliseconds"; counters.Add(latencyData); Create("NetworkCategory", "Network performance metrics", PerformanceCounterCategoryType.SingleInstance, counters); } latencyCounter = new("NetworkCategory", "Latency", false); latencyCounter.ReadOnly = false; } public void RecordLatency(long milliseconds) { latencyCounter.RawValue = milliseconds; }}
```



<https://www.linkedin.com/in/mhramini/>  
توجه: کد قبلی فقط در ویندوز پشتیبانی می‌شود.  
<https://github.com/MohamadHoseinRoohiAmini>

با ایجاد و نظرات بر این معیارهای عملکرد، توسعه‌دهندگان می‌توانند بینش ارزشمندی در مورد کارایی و قابلیت اطمینان برنامه‌های شبکه خود به دست آورند. این رویکرد پیشگیرانه امکان شناسایی و حل به موقع مشکلات عملکرد را فراهم می‌کند و اطمینان می‌دهد که برنامه‌ها در شرایط مختلف شبکه پاسخگو و قوی باقی

# شناسایی گلوگاهها

## آزمایش استرس و بار

انجام آزمایش استرس و آزمایش بار برای مشاهده چگونگی رفتار برنامه تحت شرایط مختلف شبکه ضروری است. ابزارهایی مانند Microsoft Visual Studio و Apache JMeter می‌توانند سناریوهای Load Test ترافیک بالا را شبیه‌سازی کرده و عملکرد برنامه را اندازه‌گیری کنند. مشاهده چگونگی مدیریت بار افزایش یافته توسط برنامه به شما امکان می‌دهد گلوگاههایی را که ممکن است تحت شرایط استفاده عادی آشکار نباشند، شناسایی کنید.

## ثبت وقایع و ردیابی

رویکرد مؤثر دیگر استفاده از ثبت وقایع و ردیابی برای نظارت بر فعالیت شبکه است. با پیاده‌سازی ثبت وقایع دقیق در کد شبکه خود، می‌توانید الگوها و ناهنجاری‌ها در ترافیک شبکه را آشکار کنید. به عنوان مثال، می‌توانید زمان صرف شده برای هر درخواست و پاسخ شبکه را ثبت کنید و سپس گزارش‌ها را برای شناسایی تأخیرهای غیرمعمول طولانی تحلیل کنید.

## پروفایل عملکرد

اولین گام در شناسایی گلوگاهها، پروفایل عملکرد دقیق است. همانطور که نشان داده‌ایم، ابزارهایی مانند Visual Studio، JetBrains Performance Profiler و dotTrace می‌توانند بینش دقیقی از عملکرد برنامه شما در شرایط مختلف ارائه دهند. با تحلیل استفاده از CPU، تخصیص حافظه و عملیات I/O، این ابزارها به شناسایی مناطقی که برنامه زمان یا منابع بیش از حد صرف می‌کند، کمک می‌کنند.

# استراتژی‌های بهینه‌سازی عملکرد شبکه

در چشم‌انداز همیشه در حال تکامل توسعه برنامه‌های شبکه، بهینه‌سازی عملکرد یک وظیفه حیاتی است که مستقیماً بر تجربه کاربر و کارایی عملیاتی تأثیر می‌گذارد. این بخش، "استراتژی‌های بهینه‌سازی عملکرد شبکه"، به دنبال ارائه تکنیک‌های عملی و بهترین شیوه‌ها به توسعه‌دهندگان برای افزایش عملکرد برنامه‌های شبکه‌ای خود در C# و .NET است.



## الگوهای بهینه‌سازی

ما چگونگی اعمال مدل‌های برنامه‌نویسی ناهمگام با استفاده از کلیدواژه‌های `async` و `await`، بهینه‌سازی انتقال داده با فرمتهای سریالی‌سازی کارآمد و استفاده از الگوهای بهبود عملکرد مانند کش کردن و استخراج بررسی خواهیم کرد.



## مکانیزم‌های کش

استفاده از مکانیزم‌های کش، استخراج اتصالات و تعادل بار می‌تواند به مدیریت مؤثرer منابع شبکه کمک کرده و عملکرد پایدار را تضمین کند.



## بهینه‌سازی سطح کد

در سطح کد، تکنیک‌هایی مانند برنامه‌نویسی ناهمگام، سریالی‌سازی کارآمد داده‌ها و استفاده هوشمندانه از چندنخی می‌تواند به طور قابل توجهی تأخیر را کاهش داده و توان عملیاتی را بهبود بخشد.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# بهینهسازی انتقال داده

بهینهسازی انتقال داده برای افزایش عملکرد و کارایی برنامه‌های شبکه در C# و .NET بسیار مهم است. انتقال داده کارآمد تأخیر را کاهش می‌دهد، استفاده از پهنای باند را به حداقل می‌رساند و پاسخگویی برنامه را بهبود می‌بخشد. این بخش تکنیک‌های مهم برای بهینهسازی انتقال داده، از جمله فشردهسازی، فرمتهای سریالی‌سازی کارآمد و دسته‌بندی درخواست‌ها را بررسی می‌کند.

## فشردهسازی داده

یک روش مؤثر برای بهینهسازی انتقال داده استفاده از فشردهسازی داده است. فشردهسازی اندازه داده‌های در حال انتقال را کاهش می‌دهد، که می‌تواند به طور قابل توجهی زمان مورد نیاز برای انتقال داده را کاهش داده و مصرف پهنای باند را کاهش دهد. پشتیبانی داخلی برای فشردهسازی از طریق کلاس‌هایی مانند `BrotliStream` و `GZipStream` ارائه می‌دهد.

```
using System.IO.Compression;public class BrotliDataCompressor{ public static byte[] CompressData(byte[] data) { using var outputStream = new MemoryStream(); using var brotliStream = new BrotliStream(outputStream, CompressionMode.Compress); brotliStream.Write(data, 0, data.Length); return outputStream.ToArray(); } public static byte[] DecompressData(byte[] compressedData) { using var inputStream = new MemoryStream(compressedData); using var brotliStream = new BrotliStream(inputStream, CompressionMode.Decompress); using var outputStream = new MemoryStream(); brotliStream.CopyTo(outputStream); return outputStream.ToArray(); }}
```

# سربالی‌سازی کارآمد و دسته‌بندی درخواست‌ها

## دسته‌بندی درخواست‌ها

دسته‌بندی درخواست‌ها استراتژی دیگری برای بهینه‌سازی انتقال داده است. به جای ارسال رایجی هستند، اما می‌توانند برای مجموعه داده‌های بزرگ طولانی و ناکارآمد باشند. فرمتهای سربالی‌سازی باینری، مانند Protocol Buffers یا MessagePack، سربالی‌سازی فشرده‌تر و سریع‌تری ارائه می‌دهند، که آنها را برای برنامه‌های حساس به عملکرد ایده‌آل می‌سازد.

```
using System.Text;public class DataBatcher{ private readonly  
    HttpClient _httpClient = new(); public async Task  
SendBatchedRequestsAsync(List dataItems) { var batchedData =  
        string.Join(",", dataItems); var content = new  
StringContent(batchedData, Encoding.UTF8, "application/json");  
await _httpClient.PostAsync("https://example.com/api/data",  
        content); }}
```

## فرمتهای سربالی‌سازی کارآمد

تکنیک ضروری دیگر استفاده از فرمتهای سربالی‌سازی کارآمد است JSON و XML فرمتهای سربالی‌سازی باینری، مانند Protocol Buffers یا MessagePack، سربالی‌سازی فشرده‌تر و سریع‌تری ارائه می‌دهند، که آنها را برای برنامه‌های حساس به عملکرد ایده‌آل می‌سازد.

```
using MessagePack;public class DataSerializer{ public byte[]  
    SerializeData(T data) { return  
        MessagePackSerializer.Serialize(data); } public T  
DeserializeData(byte[] serializedData) { return  
        MessagePackSerializer.Deserialize(serializedData); }}
```

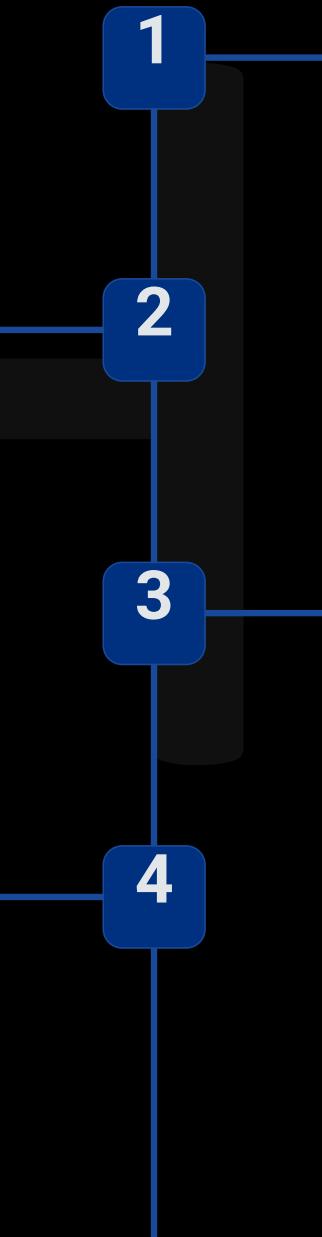
# مطالعه موردي : بهبود عملکرد برنامه شبکه

## تحلیل و راه حل

تیم از dotTrace برای تحلیل عمیق‌تر استفاده کرد و دریافت که متدهای سریالی‌سازی و پردازش داده زمان قابل توجهی مصرف می‌کنند. آنها تصمیم گرفتند از MessagePack برای سریالی‌سازی کارآمدتر استفاده کنند و عملیات پردازش داده را با استفاده از برنامه‌نویسی ناهمگام و موازی‌سازی بهینه کنند.

## نتایج

پس از پیاده‌سازی این بهینه‌سازی‌ها، برنامه کاهش ۶۰٪ در زمان پاسخ و کاهش ۴۰٪ در استفاده از پهنای باند را تجربه کرد. این بهبودها به تجربه کاربری بهتر و هزینه‌های عملیاتی کمتر منجر شد.



### شناسایی مشکل

یک برنامه تجاری با تأخیر قابل توجه در عملیات شبکه مواجه بود. با استفاده از Visual Studio Performance Profiler، تیم توسعه متوجه شد که سریالی‌سازی JSON و پردازش داده‌های بزرگ باعث گلوگاه عملکرد می‌شود.

### پیاده‌سازی

تیم کد را برای استفاده از MessagePack به جای JSON بازنویسی کرد و عملیات پردازش داده را به وظایف ناهمگام تبدیل کرد. آنها همچنین مکانیزم کش را برای کاهش فراخوانی‌های شبکه غیرضروری پیاده‌سازی کردند.

# خلاصه: بهینهسازی عملکرد شبکه

## درک و تحلیل

در فصل "بهینهسازی عملکرد شبکه" کتاب C#، ما به استراتژی‌ها و تکنیک‌های عملی پرداختیم که می‌توان به راحتی برای افزایش کارایی و پاسخگویی برنامه‌های شبکه در .NET اعمال کرد. سفر ما با درک و تحلیل معیارهای عملکرد شبکه آغاز شد، که برای تشخیص و رفع مشکلات عملکرد بسیار مهم هستند.

## ابزارها و تکنیک‌ها

ما معیارهای کلیدی مانند تأخیر، توان عملیاتی و از دست دادن بسته را معرفی کردیم و نقش ابزارهایی مانند Visual Studio Performance Profiler، JetBrains dotTrace و Event Tracing for Windows (ETW) را به عنوان منابع ارزشمند برای ضبط و تحلیل این معیارها، که به توسعه‌دهندگان کمک می‌کند گلوگاه‌های عملکرد را شناسایی کنند، تأکید کردیم.

## نتیجه‌گیری

این فصل توسعه‌دهندگان را با مجموعه ابزار جامعی برای بهینهسازی عملکرد شبکه در برنامه‌های C# و .NET مجهر کرد. با ترکیب تحلیل عملکرد دقیق با تکنیک‌های بهینهسازی هدفمند و بهترین شیوه‌ها، توسعه‌دهندگان می‌توانند اطمینان حاصل کنند که برنامه‌های آنها کارآمد، مقیاس‌پذیر و پاسخگو هستند. این استراتژی‌ها برای ارائه برنامه‌های شبکه با کیفیت بالا که نیازهای کاربران و محیط‌های امروزی را برآورده می‌کنند، حیاتی هستند.

## بهینهسازی انتقال داده

سپس فصل به تکنیک‌های عملی برای بهینهسازی انتقال داده پرداخت. ما اهمیت فشرده‌سازی داده، فرمتهای سریالی‌سازی کارآمد و دسته‌بندی درخواست‌ها را برای به حداقل رساندن تأخیر و کاهش استفاده از پهنای باند مورد بحث قرار دادیم. مثال‌های کد ساده نحوه پیاده‌سازی این تکنیک‌ها را نشان داد و به وضوح تأثیر قابل توجه آنها در بهبود عملکرد شبکه را نشان داد.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# کار با REST API ها

این ارائه به بررسی جامع مفاهیم و پیاده‌سازی REST API ها در محیط .NET 8 و #C12 می‌پردازد. ما اصول HTTP، طراحی منابع RESTful، عملیات CRUD، و بهترین شیوه‌های توسعه API را بررسی خواهیم کرد.



# فهرست مطالب

## اجزای HTTP

- افعال HTTP
- هدراهای HTTP
- کدهای وضعیت HTTP
- پیام‌های HTTP و تبادل داده

## مقدمه و مفاهیم پایه

- کار با REST API‌ها
- مقدمه‌ای بر HTTP و REST
- بررسی پروتکل HTTP

## طراحی و پیاده‌سازی عملی

- راه‌اندازی ASP.NET Core 8 Web API
- طراحی منابع RESTful
- پیاده‌سازی عملیات CRUD
- کار با داده و Entity Framework Core
- تست و اشکال‌زدایی REST API‌ها

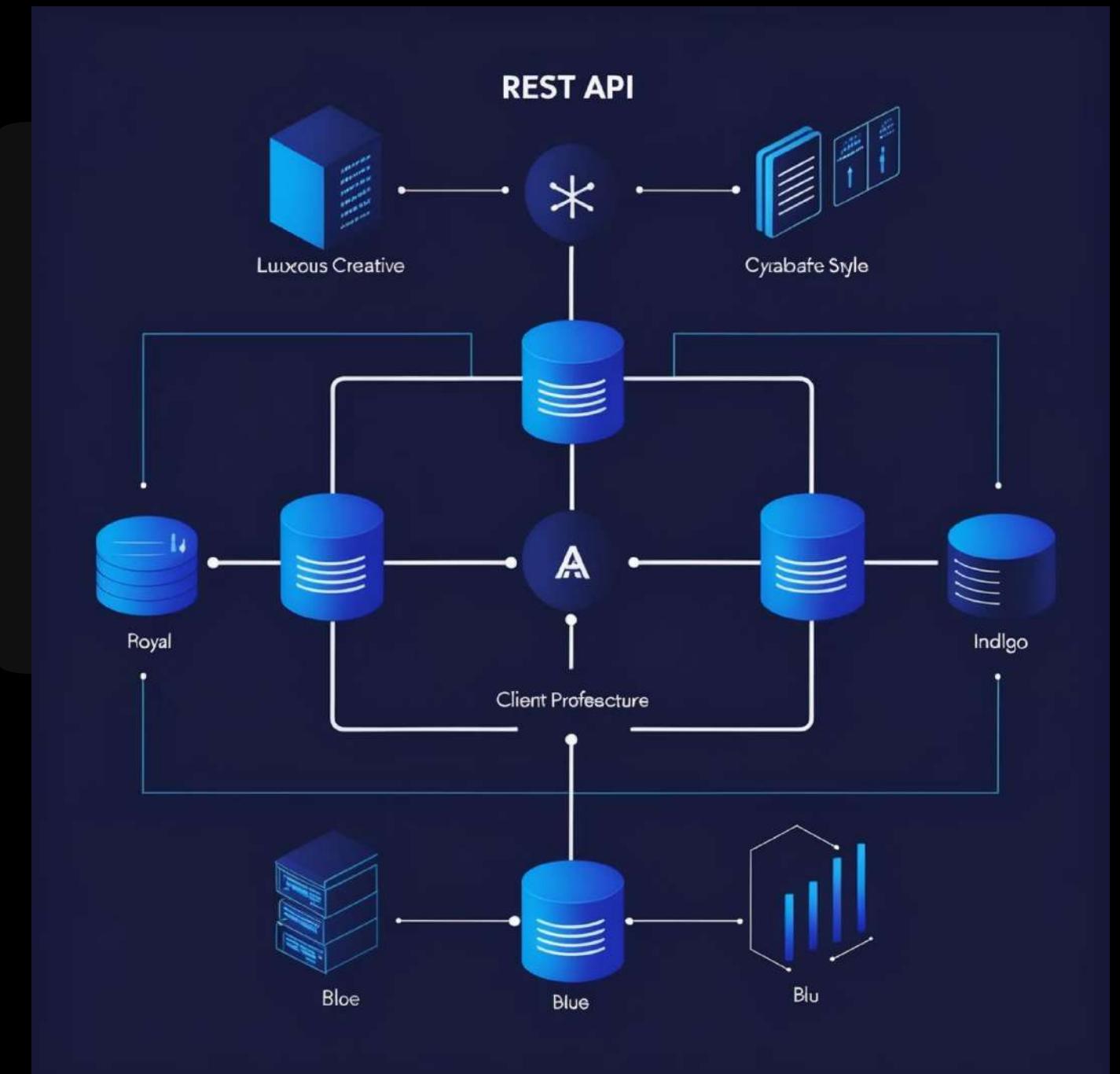
## طراحی و پیاده‌سازی

- درک REST: اصول و مفاهیم
- منابع RESTful و URI‌ها
- REST و متدهای HTTP
- بهترین شیوه‌های طراحی API

# کار با REST API ها

کار با REST API ها صرفاً یک روند نیست، بلکه جنبه‌ای اساسی از برنامه‌نویسی مدرن شبکه در .NET 8 و #C 12 است. این فناوری امکان توسعه سرویس‌های وب مقیاس‌پذیر، قابل نگهداری و قابل همکاری را فراهم می‌کند.

برای تسهیل ارتباط بین کلاینت و HTTP یا انتقال حالت نمایشی، یک سبک معماری است که از پروتکل REST مجموعه‌ای از محدودیت‌ها را تعریف می‌کنند که طراحی سرویس‌های RESTful API. سرور استفاده می‌کند و وب را هدایت می‌کنند و اطمینان می‌دهند که آنها بدون حالت، قابل ذخیره‌سازی و قادر به پشتیبانی از یک رابط یکنواخت هستند.



<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

# ستاره نمایش: ASP.NET Core

## ادغام با ویژگی‌های مدرن C#

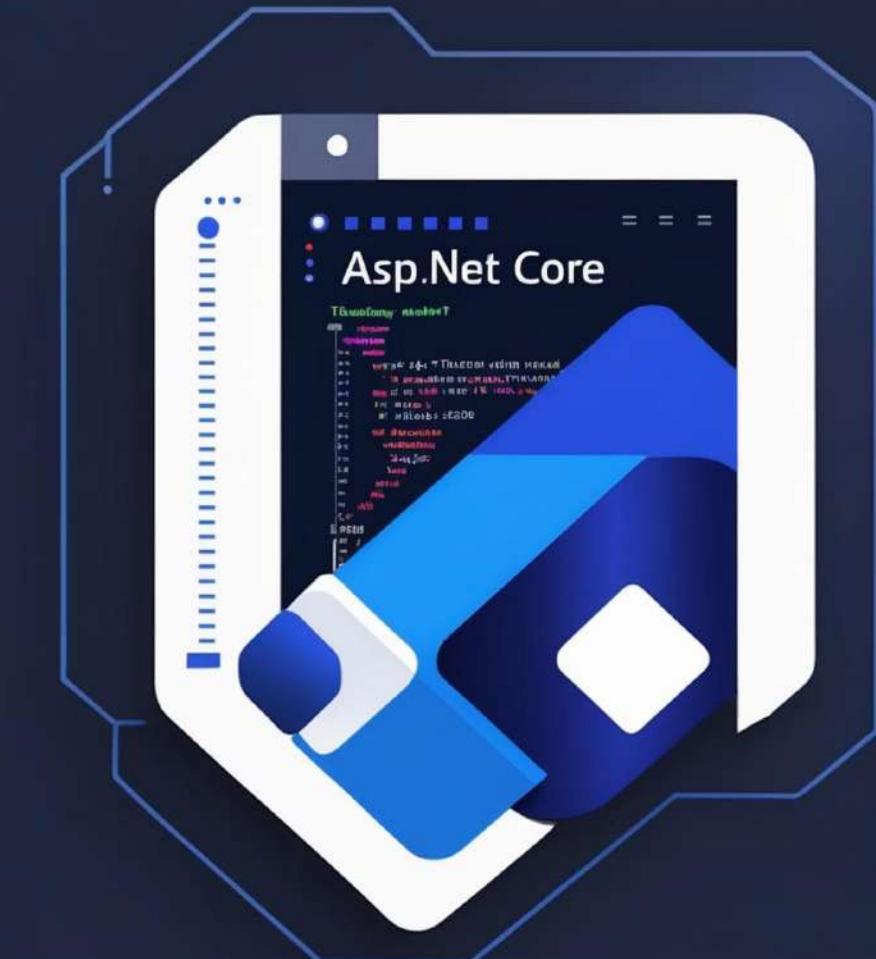
ادغام بی‌نقص آن با ویژگی‌های مدرن C#، مانند رکوردها، تطبیق الگو و API‌های حداقلی، به توسعه‌دهندگان امکان می‌دهد کدهای مختصر و خوانا بنویسند بدون اینکه در عملکرد و مقیاس‌پذیری مصالحه کنند.

## قدرت ASP.NET Core

هنگامی که صحبت از توسعه API‌های RESTful در .NET و C# می‌شود، ASP.NET Core سtarه نمایش است. این چارچوب قدرتمند و انعطاف‌پذیر تمام ابزارهای لازم برای ساخت سرویس‌های وب قوی را فراهم می‌کند.

## پشتیبانی داخلی

پشتیبانی داخلی چارچوب از متدهای HTTP، مسیریابی و اتصال مدل، فرآیند تعريف و ارائه نقاط پایانی RESTful را ساده می‌کند و توسعه‌دهندگان را آزاد می‌گذارد تا بر پیاده‌سازی منطق کسب و کار تمرکز کنند.



<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# تکامل .NET و C#

با تکامل .NET و C#، آنها بهبودهایی را به همراه می‌آورند که توسعه API‌های RESTful را بیشتر بهینه می‌کنند. ویژگی‌هایی مانند پشتیبانی از HTTP/3، گزینه‌های سریالی‌سازی بهبود یافته و مکانیسم‌های امنیتی پیشرفته اطمینان می‌دهند که برنامه‌های ساخته شده روی .NET 8 نه تنها سریع و کارآمد هستند، بلکه ایمن و آینده‌نگر نیز هستند.



<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

# مقدمه‌ای بر REST و HTTP

درک عناصر اساسی ارتباط وب نه تنها مهم، بلکه برای ساخت برنامه‌های شبکه‌ای مدرن حیاتی است. در قلب این ارتباط، پروتکل انتقال ابرمن (HTTP) قرار دارد، پروتکلی که بر نحوه تبادل داده‌ها در سراسر وب حاکم است.

قوانين HTTP برای ساختاردهی درخواست‌ها و پاسخ‌ها بین کلاینت‌ها و سوروها، ستون فقرات اینترنت هستند و بازیابی منابع و تعامل با سرویس‌ها را امکان‌پذیر می‌کنند. تسلط بر مکانیک HTTP گامی محوری برای هر توسعه‌دهنده‌ای است که با برنامه‌نویسی شبکه کار می‌کند.



<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

# بررسی پروتکل HTTP



# متد های HTTP

## GET

برای بازیابی داده از سرور بدون تغییر آن طراحی شده است. این و همگامپذیر است، به این معنی که چندین درخواست یکسان نتیجه یکسانی را بدون اثرات جانبی برمی‌گرداند.

## HEAD

مشابه GET است اما با یک تفاوت کلیدی: فقط هدرهای یک منبع را بازیابی می‌کند، نه بدنه را. این HEAD را برای بررسی متاداده‌ها مفید می‌کند.

## DELETE

یک متد ساده برای حذف منابع از سرور است. مانند PUT، DELETE همگامپذیر است، به این معنی که چندین درخواست همان اثر را خواهند داشت.



## POST

برای ارسال داده به سرور استفاده می‌شود، معمولاً برای ایجاد یک منبع جدید. این متد همگامپذیر نیست؛ هر درخواست می‌تواند منجر به نتایج متفاوتی شود، مانند ایجاد چندین ورودی در پایگاه داده.

## PUT

یک منبع موجود را با داده‌های جدید جایگزین می‌کند، آن را همگامپذیر می‌کند زیرا درخواست‌های تکراری نتیجه یکسانی تولید می‌کنند.

## PATCH

برای بهروزرسانی‌های جزئی استفاده می‌شود، جایی که فقط زیرمجموعه‌ای از منبع اصلاح می‌شود. هنگام کار با مجموعه‌های داده بزرگ مفید است.

# هدرهای HTTP

هدرهای HTTP اجزای حیاتی ارتباط بین کلاینت‌ها و سرورها هستند. آنها متادیتایی را فراهم می‌کنند که به مدیریت فرآیند درخواست و پاسخ کمک می‌کند. بدون هدرها، فرآیند ارتباط به طور قابل توجهی محدود می‌شود، زیرا آنها حاوی جفت‌های کلید-مقدار هستند که اطلاعات اضافی درباره درخواست یا پاسخ را منتقل می‌کنند.

## هدرهای درخواست

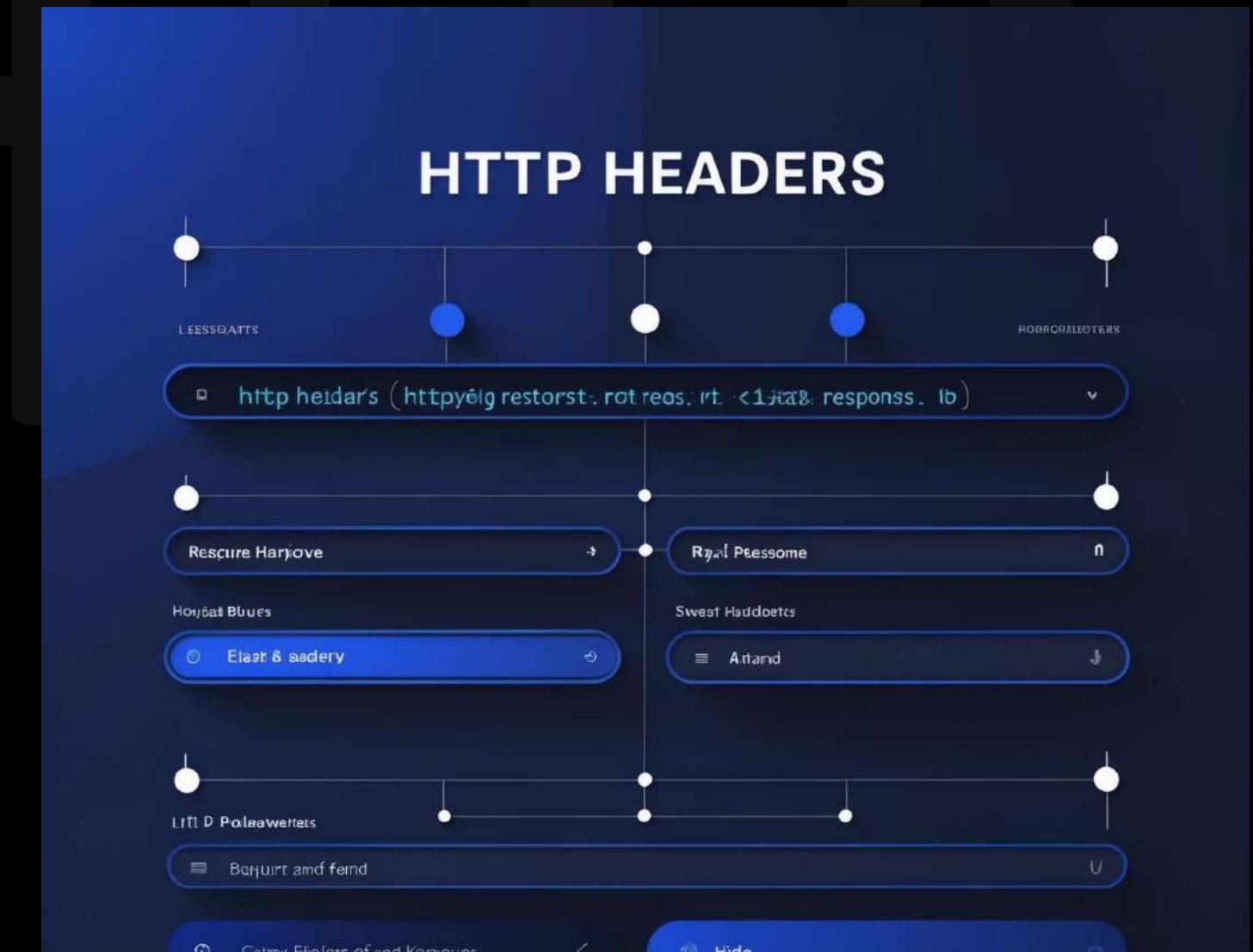
- **Accept:** نوع محتوایی که کلاینت در پاسخ انتظار دارد را مشخص می‌کند.
- **Authorization:** اعتبارنامه‌ها را برای احراز هویت ارسال می‌کند.
- **User-Agent:** برنامه کلاینت یا مرورگری که درخواست را انجام می‌دهد را شناسایی می‌کند.

- **Content-Type:** محتوای برگشتی را نشان می‌دهد نوع MIME.
- **Cache-Control:** چگونگی ذخیره‌سازی پاسخ توسط کلاینت را دیکته می‌کند.
- **Set-Cookie:** اطلاعات جلسه را در کلاینت ذخیره می‌کند.

## هدرهای پاسخ

- **Strict-Transport-Security:** را اجباری می‌کند HTTPS استفاده از.
- **Content-Security-Policy:** کمک می‌کند XSS به جلوگیری از حملات.

## هدرهای مرتبط با امنیت



<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

# کدهای وضعیت HTTP

## کدهای اطلاعاتی (1xx)

1

این کدها نشان می‌دهند که درخواست دریافت شده و پردازش ادامه دارد.

- 100 Continue: بخش اولیه درخواست دریافت شده و کلاینت باید به ارسال ادامه دهد:

## کدهای موفقیت (2xx)

2

این کدها نشان می‌دهند که درخواست با موفقیت دریافت، درک و پذیرفته شده است.

- درخواست موفق بوده است: 200 OK

درخواست منجر به ایجاد یک منبع جدید شده است: 201 Created:

- سرور درخواست را پردازش کرده اما محتوایی برای برگرداندن ندارد: 204 No Content:

## کدهای تغییر مسیر (3xx)

3

این کدها نشان می‌دهند که اقدامات بیشتری برای تکمیل درخواست لازم است.

جدیدی منتقل شده است URL منبع درخواست شده به طور دائم به: 301 Moved Permanently:

- دیگری در دسترس است URL منبع به طور موقت در: 302 Found:

## کدهای خطای کلاینت (4xx)

4

این کدها نشان می‌دهند که درخواست حاوی نحو نادرست است یا نمی‌تواند انجام شود.

سرور نمی‌تواند درخواست را به دلیل نحو نامعتبر درک کند: 400 Bad Request:

احراز هویت لازم است: 401 Unauthorized:

سرور درخواست را درک می‌کند اما از مجاز کردن آن خودداری می‌کند: 403 Forbidden:

منبع درخواست شده در سرور یافت نشد: 404 Not Found:

## کدهای خطای سرور (5xx)

5

این کدها نشان می‌دهند که سرور در انجام یک درخواست معتبر شکست خورده است.

یک خطای عمومی هنگامی که شرایط غیرمنتظره با آن مواجه می‌شود: 500 Internal Server Error:

سرور، به عنوان یک دروازه یا پروکسی، پاسخ نامعتبری از سرور بالادست دریافت کرد: 502 Bad Gateway:

سرور موقتاً قادر به پردازش درخواست نیست: 503 Service Unavailable:

<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

# پیام‌های HTTP و تبادل داده

## پیام پاسخ HTTP

در ارتباط HTTP، پیام‌ها اساس تبادل داده بین کلاینت‌ها و سرورها هستند. هر تعامل HTTP شامل یک درخواست ارسال شده توسط کلاینت و یک پاسخ برگردانده شده توسط سرور است. این انواع پیام به شکل خاصی ساختاربندی شده‌اند، متشکل از یک خط شروع، هدرها و یک بدن.

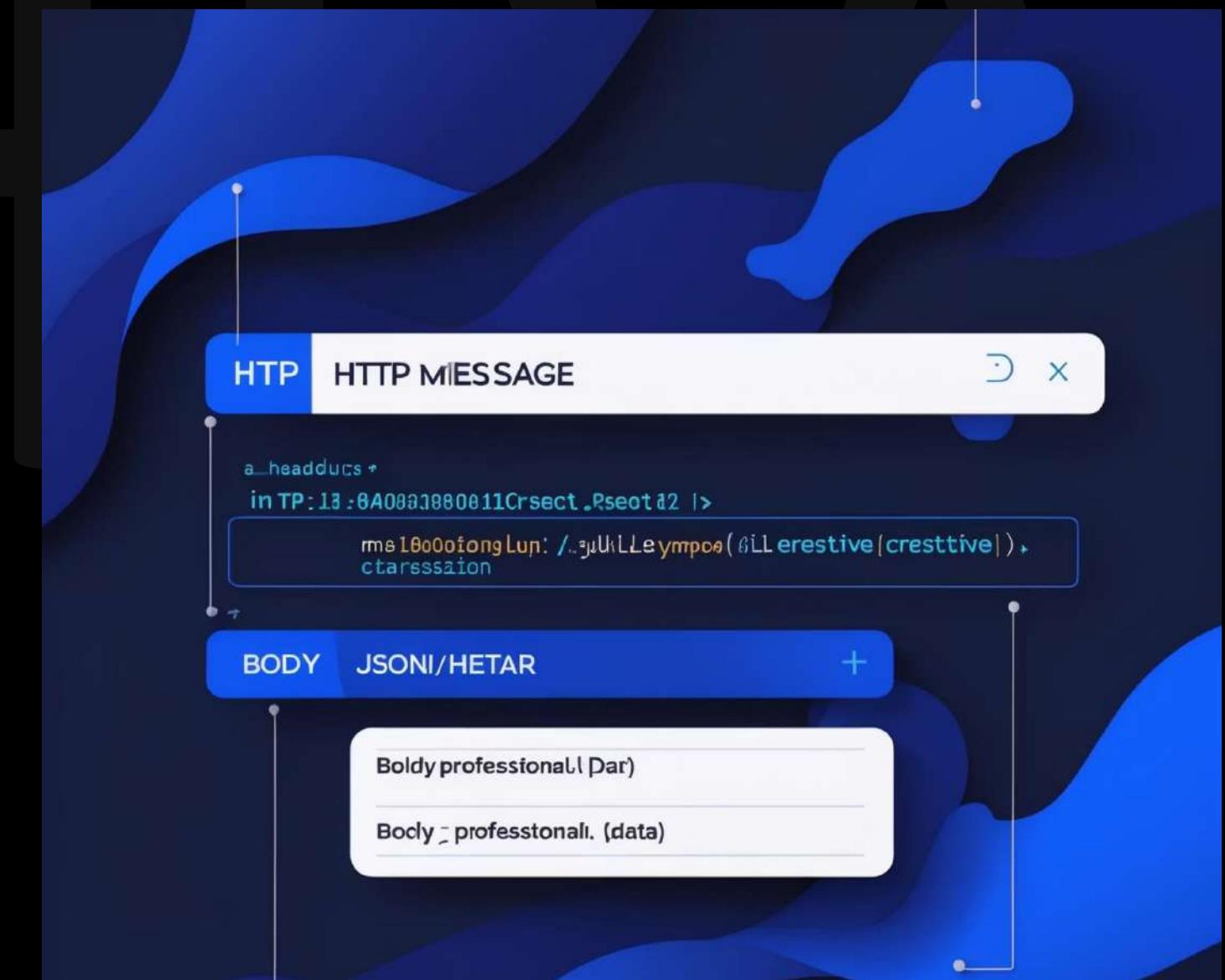
به طور مشابه، یک پیام پاسخ HTTP‌ها یک خط وضعیت شروع می‌شود، که شامل نسخه پروتکل، یک کد وضعیت و یک پیام وضعیت است. پس از خط وضعیت، هدرهای پاسخ اطلاعات اضافی درباره پاسخ را فراهم می‌کنند. بدن پاسخ حاوی داده‌های واقعی است.

## پیام درخواست HTTP

یک پیام درخواست HTTP با یک خط درخواست شروع می‌شود که حاوی متد (GET یا POST) مانند URI را هدف و نسخه پروتکل است. پس از خط درخواست، هدرها زمینه اضافی درباره درخواست را فراهم می‌کنند. بدنه اختیاری درخواست حاوی داده‌های واقعی در حال انتقال است.

## فرمت‌های تبادل داده

یکی از فرمتهای رایج مورد استفاده در برنامه‌های وب مدرن برای تبادل داده ساختاریافته است (نشانه‌گذاری شیء جاوا اسکریپت) JSON. سبک، آسان برای تجزیه و به طور گسترده در زبان‌های برنامه‌نویسی مختلف پشتیبانی می‌شود. فرمت دیگری (زبان نشانه‌گذاری قابل گسترش) XML های جدید کمتر رایج است، اگرچه امروزه در API است.



# درک: REST: اصول و مفاهیم

## مفهوم منبع

در قلب REST مفهوم منبع قرار دارد، که هر داده یا سرویس قابل دسترس در یک شبکه را نمایندگی می‌کند. منابع می‌توانند هر چیزی از یک پروفایل کاربر تا یک کاتالوگ محصول یا حتی یک پست وبلاگ باشند. هر منبع به طور منحصر به فرد توسط یک شناسه منبع یکنواخت (URI) شناسایی می‌شود، که به عنوان آدرس آن در وب عمل می‌کند.

## معماری توزیع شده

یک سبک معماری طراحی شده برای سیستم‌های توزیع شده، به REST این سبک مجموعه‌ای از محدودیت‌ها و ویژه سرویس‌های وب است. اصول را تعریف می‌کند که نحوه تعامل این سیستم‌ها را هدایت می‌کند، مانند متدها، کدهای وضعیت و هدرها، از HTTP اجزای اساسی REST برای تسهیل ارتباط بین کلاینت‌ها و سرورها به شیوه‌ای بدون حالت و مقیاس‌پذیر استفاده می‌کند.

## جداسازی دغدغه‌ها

اصل اساسی دیگر REST، جداسازی دغدغه‌ها بین کلاینت و سرور است. در یک معماری RESTful، کلاینت رابط کاربری و تجربه کاربر را مدیریت می‌کند، در حالی که سرور منابع و داده‌ها را مدیریت می‌کند. این تقسیم‌بندی واضح به کلاینت‌ها و سرورها اجازه می‌دهد به طور مستقل تکامل یابند.

## ارتباط بدون حالت

یک ویژگی کلیدی سیستم‌های RESTful، ارتباط بدون حالت بین کلاینت‌ها و سرورها است. هر درخواست HTTP از یک کلاینت به سرور باید تمام اطلاعات لازم برای پردازش آن توسط سرور را شامل باشد. این بدان معناست که سرور هیچ حالت جلسه خاصی را بین درخواست‌ها حفظ نمی‌کند. این ویژگی مقیاس‌پذیری و قابلیت اطمینان را افزایش می‌دهد.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# منابع RESTful و URI ها

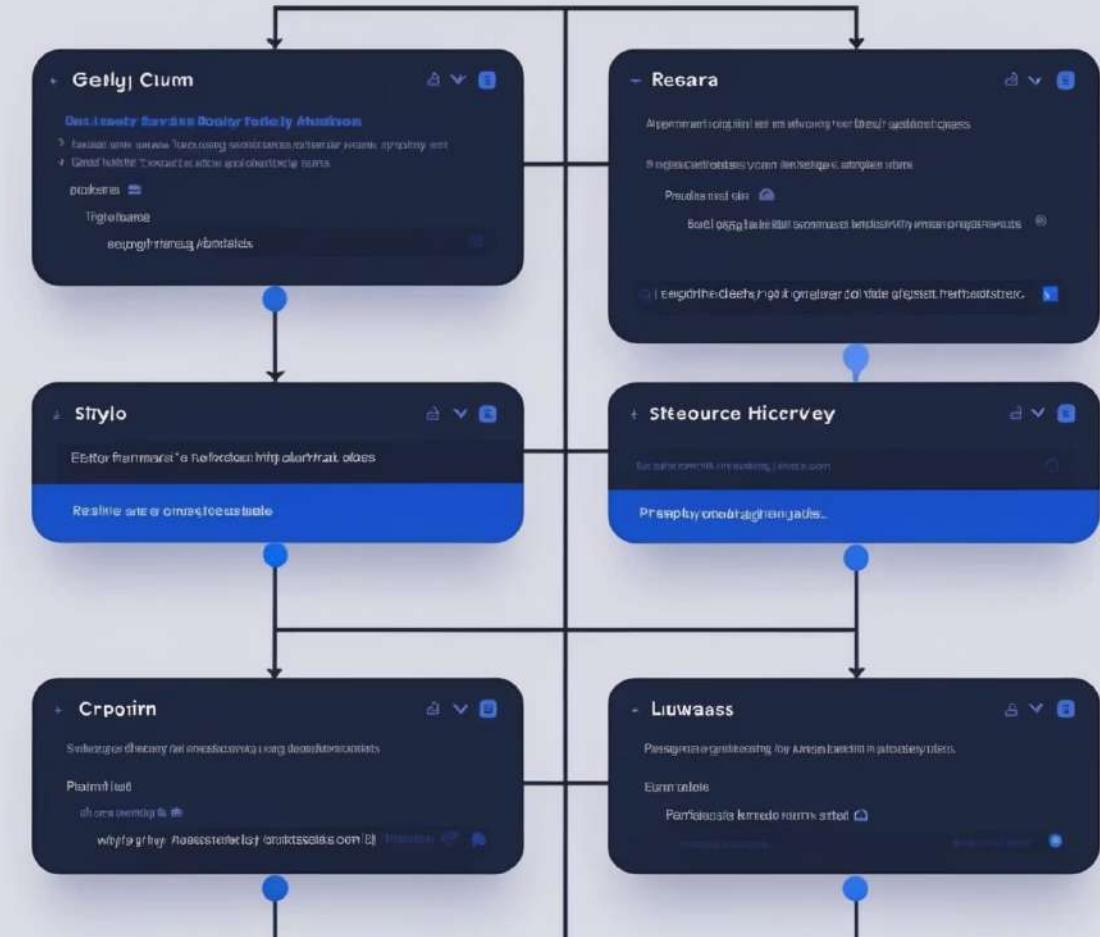
در سیستم‌های RESTful، مفهوم منابع محور چگونگی تعامل کلاینت‌ها و سرورها است. یک منبع هر موجودیت یا داده‌ای را نمایندگی می‌کند که می‌تواند از طریق وب دسترسی و دستکاری شود. این می‌تواند از یک کاربر منفرد یا مجموعه‌ای از محصولات تا یک سند فردی یا حتی یک فرآیند سمت سرور متغیر باشد.

یک شناسه منبع یکنواخت (URI) هر منبع را در یک سیستم RESTful به طور منحصر به فرد شناسایی می‌کند. ها یک راه استاندارد و قابل خواندن توسط انسان برای آدرس‌دهی منابع فراهم می‌کنند، که تعامل کلاینت‌ها با API را ساده می‌کنند.

## بهترین شیوه‌های طراحی URI

- استفاده از اسامی به جای افعال در URL ها
- حفظ سازگاری در سراسر API
- استفاده از اسامی جمع برای مجموعه‌ها
- نسخه‌بندی API در URL API
- استفاده از پارامترهای پرس‌و‌جواب برای فیلتر کردن

## Restful URI Structure Dierarchy Style



<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# REST و متدهای HTTP



## GET

برای بازیابی اطلاعات از سرور استفاده می‌شود. یک متد "ایمن" محسوب می‌شود زیرا منبع را تغییر نمی‌دهد و همگام‌پذیر است. GET برای سناریوهای بازیابی داده ایده‌آل است.

## POST

برای ارسال داده به سرور استفاده می‌شود، معمولاً برای ایجاد یک منبع جدید. این متد همگام‌پذیر نیست؛ هر درخواست می‌تواند منجر به نتایج متفاوتی شود. هنگام انجام عملیاتی که حالت سرور را تغییر می‌دهند ضروری است.



## PUT/PATCH

هر دو برای بهروزرسانی منابع موجود استفاده می‌شوند اما در دامنه متفاوت هستند. PUT یک منبع کامل را با یک منبع جدید جایگزین می‌کند، در حالی که PATCH برای بهروزرسانی‌های جزئی استفاده می‌شود، جایی که فقط زیرمجموعه‌ای از منبع اصلاح می‌شود.



## DELETE

برای حذف منابع از سرور استفاده می‌شود. مانند DELETE، همگام‌پذیر است، به این معنی که چندین درخواست DELETE برای همان منبع همان اثر را خواهند داشت. برای مدیریت چرخه عمر منبع در یک API RESTful ضروری است.

# بهترین شیوه‌های طراحی API RESTful

## سازگاری در قراردادهای نام‌گذاری

استفاده از اسمی جمع برای مجموعه‌ها، مانند `/api/users/`، و اسمی مفرد هنگام اشاره به منابع فردی، مانند `/api/users/123/`، وضوح API را افزایش می‌دهد. همچنین، اجتناب از افعال در URI‌ها به حفظ تمرکز بر منابع کمک می‌کند.

## منبع-محور باشد

منابع، مانند کاربران، محصولات یا سفارشات، باید نقطه کانونی API باشند، هر کدام با یک شناسه منبع یکنواخت (URI) منحصر به فرد شناسایی می‌شوند. اقدامات API روی این منبع باید توسط متدهای HTTP هدایت شوند.

## مدیریت خطا

هنگامی که مشکلی پیش می‌آید، API باید پیام‌های خطای واضح و آموزنده ارائه دهد که به کلاینت‌ها کمک کند بفهمند چه اتفاقی افتاده و چگونه آن را برطرف کنند. استفاده از کدهای وضعیت HTTP استاندارد اطمینان می‌دهد که کلاینت‌ها می‌توانند پاسخ API را به راحتی تفسیر کنند.

## نسخه‌بندی

تغییرات در یک API اجتناب‌ناپذیر است، اما این تغییرات باید اتکای کلاینت‌ها به نسخه‌های قبلی را حفظ کنند. با گنجاندن نسخه API در URI، کلاینت‌ها می‌توانند به تعامل با نسخه فعلی ادامه دهند.

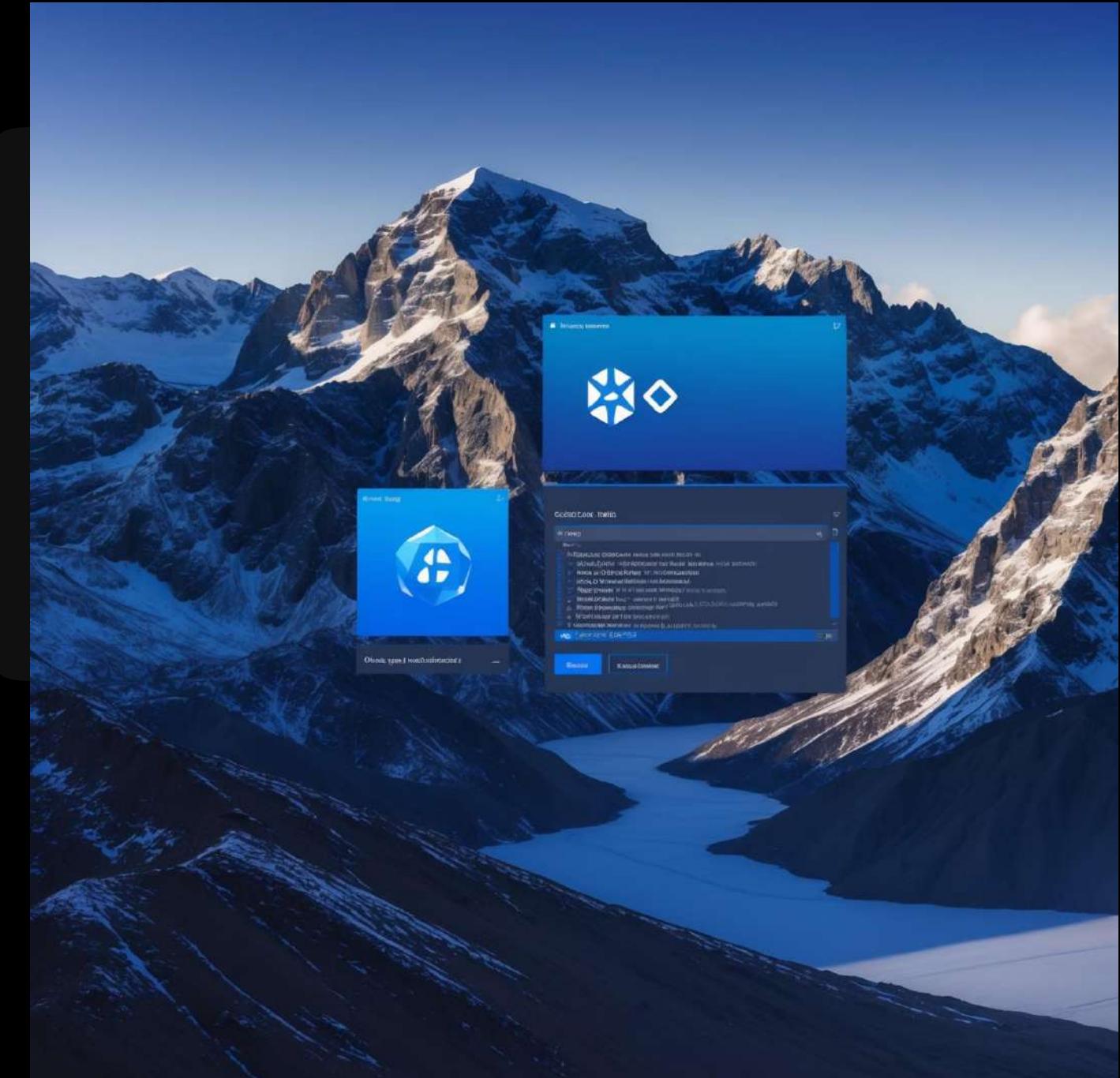
<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# راهاندازی ASP.NET Core 8 Web API

برای شروع ساخت یک ASP.NET Core 8 API RESTful، اولین قدم راهاندازی یک پروژه Web API جدید با استفاده از Visual Studio 2022 Community Edition است. یک چارچوب قدرتمند برای ایجاد API‌های وب مدرن و مقیاس‌پذیر ارائه می‌دهد که از آخرین ویژگی‌های .NET، از جمله API‌های حداقلی و تزریق وابستگی بهبود یافته، بهره می‌برد.

1. Visual Studio شروع، یک پروژه جدید ایجاد کنید Create a new project باز کنید و با انتخاب 2022 از پنجره شروع، یک پروژه جدید ایجاد کنید Create a new project را باز کنید و با انتخاب 2022 از لیست قالب‌های پروژه، ASP.NET Core Web API را انتخاب کنید و روی Next کلیک کنید.
2. یک نام برای پروژه خود ارائه دهید، مکانی برای ذخیره آن انتخاب کنید و روی Create کلیک کنید.
3. در دیالوگ بعدی، اطمینان حاصل کنید که 8 به عنوان چارچوب هدف انتخاب شده است و گزینه فعال‌سازی پشتیبانی OpenAPI برای تولید خودکار مستندات API را بررسی کنید.
4. از پنجره شروع، یک پروژه جدید ایجاد کنید Create a new project باز کنید و با انتخاب 2022 از لیست قالب‌های پروژه، ASP.NET Core Web API را انتخاب کنید و روی Next کلیک کنید.



<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

پس از تولید پروژه، Visual Studio یک ساختار پوشیده پیش‌فرض ایجاد می‌کند که شامل چندین جزء کلیدی است. پوشیده

# طراحی منابع RESTful

تعیین منابع یک گام حیاتی هنگام طراحی API‌های RESTful است. منابع موجودیت‌های کلیدی را نمایندگی می‌کنند که API‌ارائه می‌دهد و روی آنها عمل می‌کند، مانند کاربران، محصولات یا سفارشات. در یک API با ساختار خوب، هر منبع با وضوح و سازگاری مدل‌سازی می‌شود، اطمینان می‌دهد که توسعه‌دهنگان و کلاینت‌ها می‌توانند به راحتی تعامل کنند.

در ASP.NET Core 8، یک منع معمولاً توسط یک کلاس مدل نمایندگی می‌شود. به عنوان مثال، یک کلاس Product ساده را در نظر بگیرید که یک محصول را در یک سیستم تجارت الکترونیک نمایندگی می‌کند:

```
public class Product{    public int Id { get; set; }    public string Name { get; set; }    public string Description { get; set; }    public decimal Price { get; set; }}
```

پس از تعریف کلاس منبع، گام بعدی ایجاد یک کنترلر برای مدیریت نقاط پایانی API برای مدیریت این منبع است:

```
[ApiController][Route("api/[controller]")]public class ProductController : ControllerBase{    [HttpGet]    public IEnumerable GetAllProducts()    {        // Returns a list of mock products        return new List<Product>        {            new Product { Id = 1, Name = "Laptop", Description = "A high-performance laptop", Price = 999.99m },            new Product { Id = 2, Name = "Headphones", Description = "Noise-cancelling headphones", Price = 199.99m }        };    }    [HttpGet("{id}")]    public ActionResult GetProductById(int id)    {        var product = new Product { Id = id, Name = "Sample Product", Description = "Sample Description", Price = 9.99m };        return Ok(product);    }}
```

# پیاده‌سازی عملیات CRUD

2

خواندن (GET)

```
[HttpGet]public IEnumerable GetAllProducts(){    // Logic to retrieve all products (e.g., from a database)
    return new List<Product> {
        new Product { Id = 1, Name = "Laptop", Description = "A high-performance laptop", Price = 999.99m },
        new Product { Id = 2, Name = "Headphones", Description = "Noise-cancelling headphones", Price = 199.99m }
    };
}

[HttpGet("{id}")]
public ActionResult GetProductById(int id){
    var product = new Product { Id = id, Name = "Sample Product", Description = "Sample Description", Price = 9.99m };
    if (product == null) {
        return NotFound();
    }
    return Ok(product);
}
```

1

ایجاد (POST)

```
[HttpPost]public ActionResult CreateProduct(Product newProduct){    // Add logic to save the product (e.g.,
    database call)
    newProduct.Id = new Random().Next(1, 1000); // Simulating ID assignment
    return CreatedAtAction(nameof(GetProductById), new { id = newProduct.Id }, newProduct);
}
```

در این مثال، متد CreateProduct یک شیء Product را از بدن درخواست می‌پذیرد و یک ID ابه آن اختصاص می‌دهد، شبیه‌سازی یک درج پایگاه داده. متدهای CreateAtAction یک پاسخ HTTP 201 با عنوان تازه ایجاد شده برگرداند.

4

حذف (DELETE)

```
[HttpDelete("{id}")]
public IActionResult DeleteProduct(int id){
    // Logic to delete the product (e.g.,
    database call)
    return NoContent(); // Indicates the resource was successfully deleted
}
```

3

بروزرسانی (PUT)

```
[HttpPut("{id}")]
public IActionResult UpdateProduct(int id, Product updatedProduct){
    if (id != updatedProduct.Id) {
        return BadRequest("Product ID mismatch.");
    }
    // Logic to update the product (e.g., database call)
    return NoContent(); // Indicates successful update but no content returned
}
```

# کار با داده و Entity Framework Core

پس از راه اندازی اتصال پایگاه داده، گام بعدی تعریف کلاس DbContext است. این کلاس اشیاء موجودیت را در زمان اجرا مدیریت می کند، که شامل ردیابی تغییرات، حفظ روابط و انجام عملیات پایگاه داده است:

این کلاس برای کار با داده در یک ASP.NET Core 8 Web API، راه اندازی EF Core در پروژه است. این شامل نصب بسته های NuGet لازم و پیکربندی اتصال پایگاه داده است. پس از نصب، این بسته ها ابزارهایی برای تعامل با یک پایگاه داده SQL Server از طریق EF Core فراهم می کنند.

```
var builder = WebApplication.CreateBuilder(args); // Add services to the
container.builder.Services.AddControllers(); builder.Services.AddDbContext(options => options.UseSqlServer(
builder.Configuration.GetConnectionString("DefaultConnection"))); var app = builder.Build();
```

پس از راه اندازی پایگاه داده و اعمال مهاجرت ها، گام بعدی پیاده سازی عملیات داده در کنترلر API با استفاده از EF Core است. به عنوان مثال، پیاده سازی متدهای GET برای بازیابی محصولات از پایگاه داده:

```
[HttpGet]public async Task<IActionResult> GetAllProducts(){ return await _context.Products.ToListAsync();}
```



# تست و اشکال‌زدایی REST API‌ها

## تست با xUnit

است که راهی ساده و انعطاف‌پذیر برای نوشتتن. یک چارچوب تست محبوب در xUnit تست‌های واحد عملکرد اجزای فردی را. شما ارائه می‌دهد API‌تست‌های واحد و یکپارچگی برای سیستم‌های خارجی، مانند پایگاه‌های داده، را در یک سناریوی دنیای واقعی آزمایش کنید. داده را آزمایش می‌کنند.

```
public class ProductControllerTests{    private readonly ProductController _controller;    private readonly Mock _contextMock;    public ProductControllerTests() {        _contextMock = new Mock();        _controller = new ProductController(_contextMock.Object);    }    [Fact]    public void GetAllProducts_ReturnsProducts() {        // Arrange        var products = new List<Product> {            new Product { Id = 1, Name = "Laptop", Description = "A laptop", Price = 1000m },            new Product { Id = 2, Name = "Phone", Description = "A smartphone", Price = 500m }        };        _contextMock.Setup(c => c.Products.ToList()).Returns(products);        // Act        var result = _controller.GetAllProducts();        // Assert        Assert.Equal(2, result.Count());        Assert.Equal("Laptop", result.First().Name);    }}
```

## تست‌های یکپارچگی

تست‌های یکپارچگی اغلب برای سناریوهای پیچیده‌تر، مانند آزمایش ایجاد محصولات (POST)، مناسب‌تر هستند. تست‌های یکپارچگی به شما امکان می‌دهند نحوه تعامل API‌ها با سیستم‌های خارجی، مانند پایگاه‌های داده، را در یک سناریوی دنیای واقعی آزمایش کنید.

```
public class ProductApiIntegrationTests : IClassFixture<WebApplicationFactory<Program>>{    private readonly HttpClient _client;    public ProductApiIntegrationTests(WebApplicationFactory<Program> factory) {        _client = factory.CreateClient();    }    [Fact]    public async Task CreateProduct>ReturnsCreatedProduct() {        // Arrange        var newProduct = new Product { Name = "Tablet", Description = "A tablet device", Price = 300m };        var content = new StringContent(JsonConvert.SerializeObject(newProduct), Encoding.UTF8, "application/json");        // Act        var response = await _client.PostAsync("/api/products", content);        // Assert        response.EnsureSuccessStatusCode();        var responseBody = await response.Content.ReadAsStringAsync();        var createdProduct = JsonConvert.DeserializeObject(responseBody);        Assert.Equal("Tablet", createdProduct.Name);    }}
```

## اشکال‌زدایی

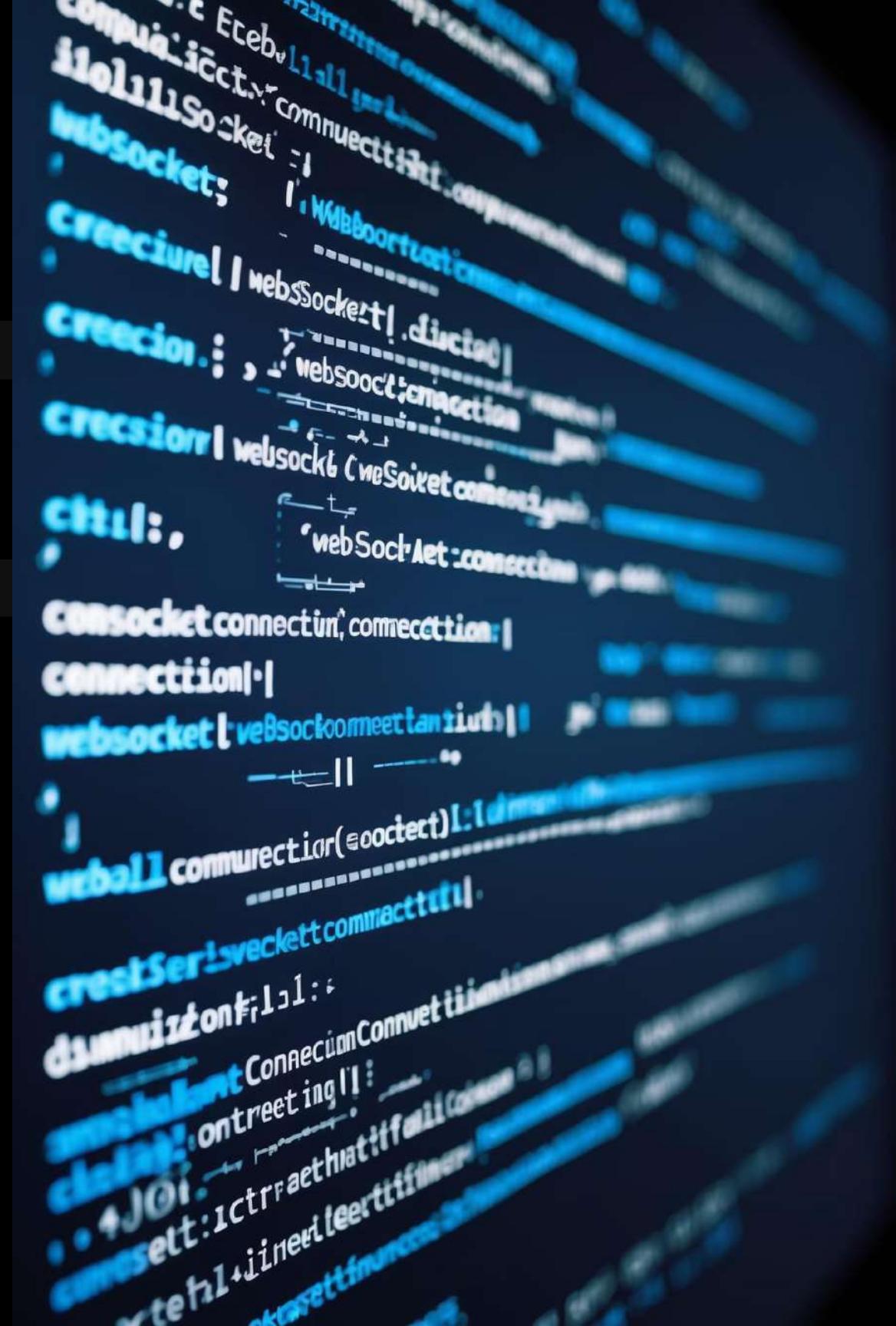
علاوه بر نوشتن تست‌ها، اشکال‌زدایی API‌ها بخش مهم دیگری از اطمینان از قابلیت اطمینان آن است 2022 Visual Studio. ابزارهای اشکال‌زدایی قدرتمندی ارائه می‌دهد که به شما امکان می‌دهد نقاط توقف تنظیم کنید، متغیرها را بررسی کنید و خط به خط اجرای کد را دنبال کنید.

در طول توسعه، مفید است که API را به صورت محلی اجرا کنید و از نقاط توقف برای بررسی رفتار نقاط پایانی خاص استفاده کنید. به عنوان مثال، می‌توانید نقاط توقف را در متد CreateProduct تنظیم کنید تا درخواست ورودی را بررسی کنید، وضعیت محصول در حال ایجاد را مشاهده کنید و تعامل پایگاه داده را تأیید کنید.

# وبسوکت‌ها در سی‌شارپ

یک راهنمای جامع برای پیاده‌سازی ارتباطات وبوسوکت در سناریوهای سرور و کلاینت با استفاده از داتنوت و سی‌شارپ

<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>



# فهرست مطالب

## مکانیزم پروتکل وب‌سوکت

نحوه عملکرد وب‌سوکت‌ها و مقایسه با HTTP سنتی

## مقدمه و مفاهیم پایه

آشنایی با وب‌سوکت‌ها و اهمیت آن‌ها در برنامه‌های شبکه

## ویژگی‌های پیشرفتی

مدیریت گروه‌های کلاینت، فشرده‌سازی و اشکال‌زدایی

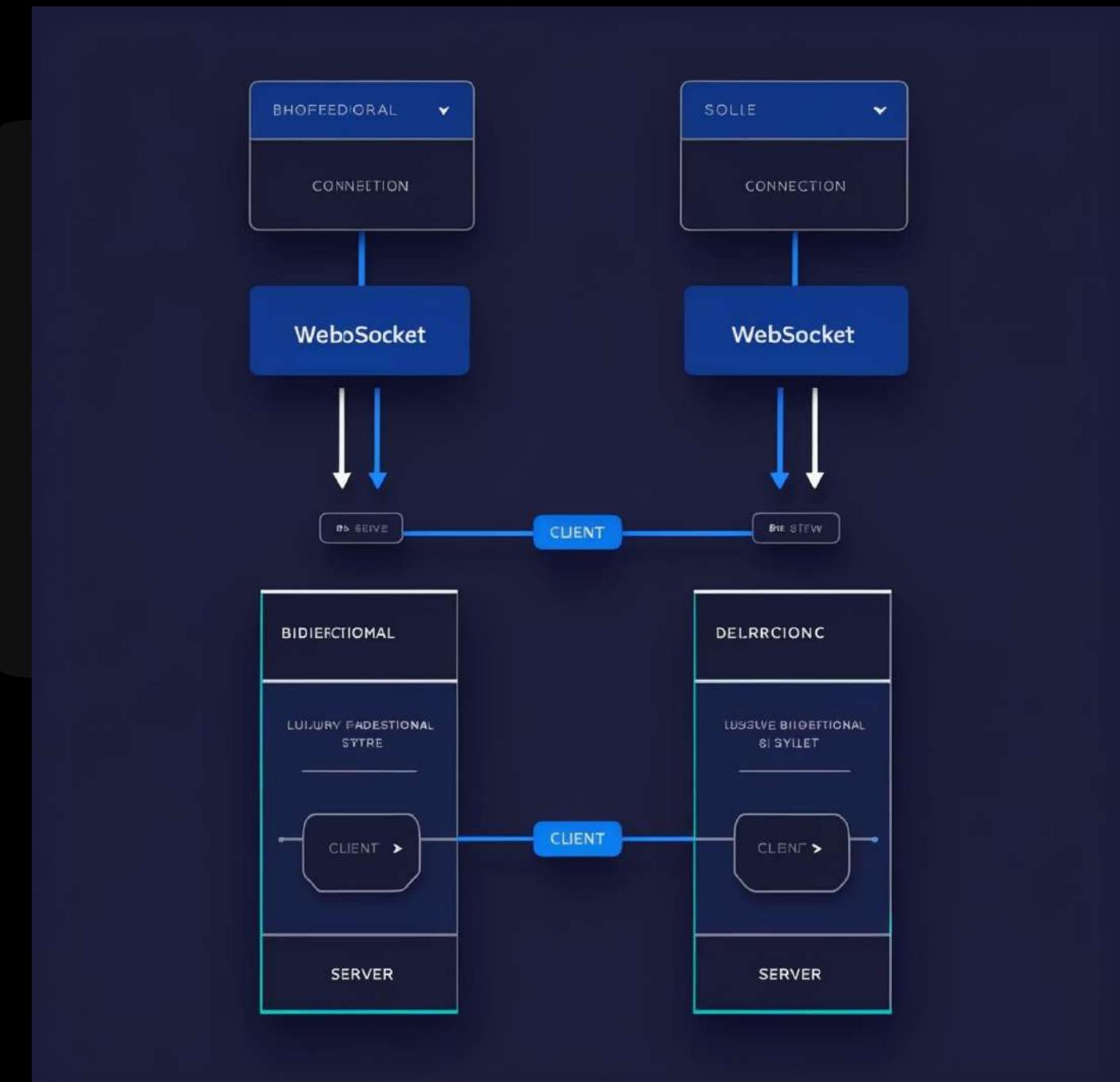
## پیاده‌سازی در سی‌شارپ

راهاندازی سرور و کلاینت وب‌سوکت با استفاده از دات‌نوت

# کار با وب‌سوکت‌ها

وب‌سوکت‌ها یک فناوری حیاتی هستند که امکان ارتباط دو طرفه و بلادرنگ بین کلاینت‌ها و سرورها را فراهم می‌کنند و تبادل داده‌های پویاتر در برنامه‌های شبکه را تسهیل می‌کنند.

برخلاف درخواست‌های HTTP استنی که از الگوی درخواست-پاسخ پیروی می‌کنند، وب‌سوکت‌ها یک اتصال پایدار ایجاد می‌کنند که در آن داده‌ها می‌توانند آزادانه در هر دو جهت جریان یابند.



<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

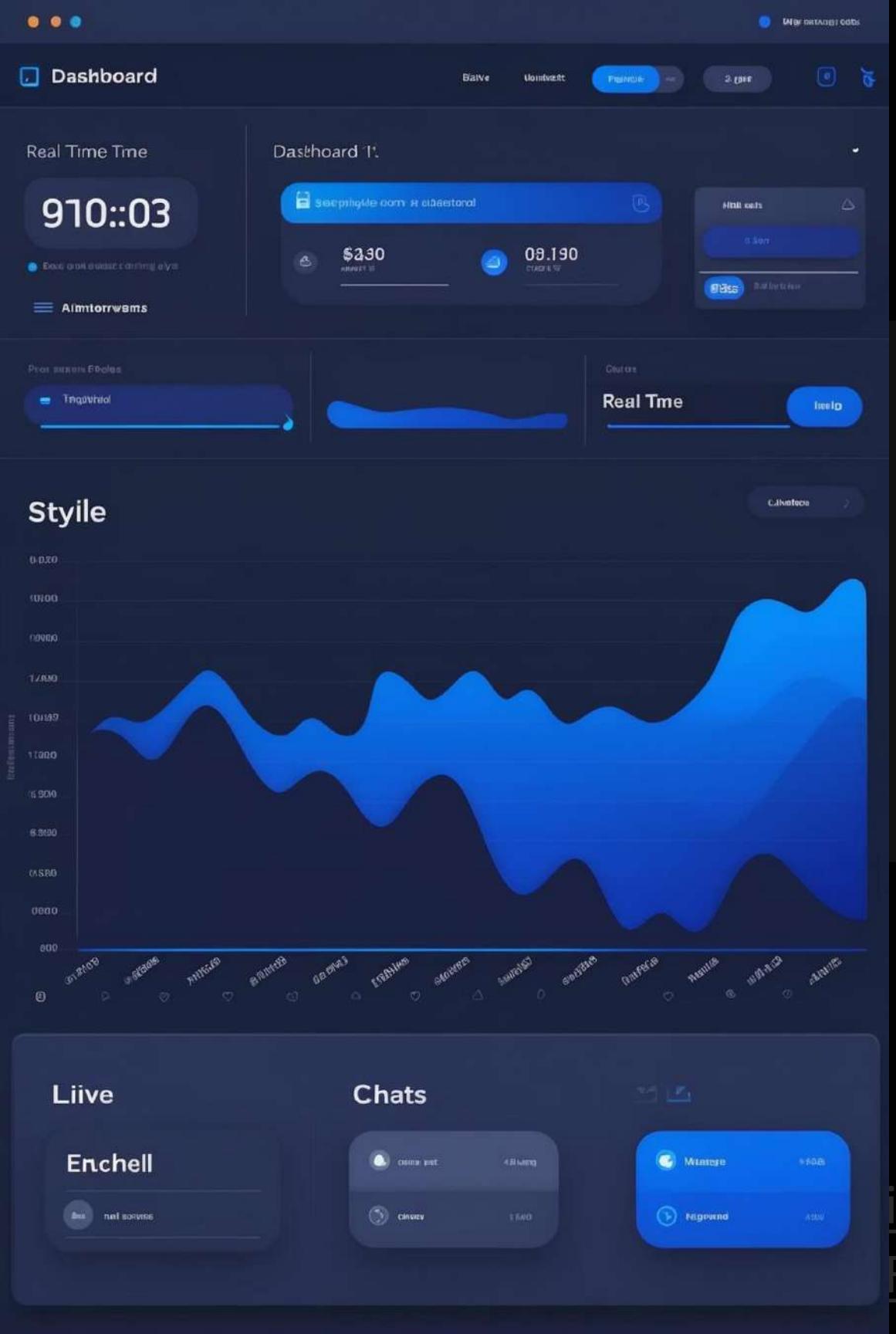
# مرور کلی پروتکل وب‌سوکت



پروتکل وب‌سوکت مبتنی بر فریم است، به این معنی که داده‌ها به صورت فریم‌های مجزا منتقل می‌شوند. این فریم‌ها می‌توانند حاوی داده‌های متنی یا باینری باشند و امکان ارتباط انعطاف‌پذیر را بسته به نیازهای برنامه فراهم می‌کنند.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>



# موارد استفاده، مزایا و مقایسه با HTTP سنتی

## موارد استفاده وبسوکت

- برنامه‌های چت و پیام‌رسانی آنلاین
- بازی‌های چندنفره آنلاین
- داشبوردهای تحلیلی بلادرنگ
- سیستم‌های معاملاتی مالی
- دستگاه‌های اینترنت اشیا (IoT)
- سیستم‌های همکاری آنلاین

## مزایا نسبت به HTTP سنتی

- ارتباط دو طرفه بدون نیاز به درخواست‌های مکرر
- کاهش تأخیر با حذف سربار اتصال مجدد
- کاهش ترافیک شبکه با حذف نیاز به polling
- امکان ارسال داده از سرور به کلاینت در هر زمان
- بهبود کارایی برای برنامه‌های بلادرنگ

وبسوکت‌ها در مواردی که ارتباط بلادرنگ مورد نیاز است، مانند چت زنده، بازی یا تحلیل‌های بلادرنگ، که هم کلاینت و هم سرور باید فوراً به رویدادها واکنش نشان دهند، واقعاً درخشان هستند.

# مکانیزم پروتکل وبسوکت و ویژگی‌های پیشرفته

## ساختار فریم

انتقال داده با استفاده از فریم‌های متنی یا  
باینری با سربار کم

## فشرده‌سازی

کاهش اندازه پیام‌ها برای بهبود کارایی در  
محیط‌های با پهنای باند محدود

## مدیریت گروه کلاینت

ایجاد تعاملات هدفمند مانند اتاق‌های چت یا  
لابی‌های بازی



## دست‌دهی اولیه

شروع با یک درخواست HTTP برای ارتقا به  
پروتکل وبسوکت

## Ping-Pong های پیام

حفظ اتصال‌های فعال و تشخیص قطع ارتباط

این ویژگی‌ها به ایجاد یک سیستم ارتباطی بلادرنگ مقیاس‌پذیر و قوی کمک می‌کنند و وеб‌سوکت‌ها را به عنوان راه حل برتر برای برنامه‌های شبکه مدرن متمایز می‌کنند.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# ملاحظات عملی

2

## مدیریت قطع ارتباط شبکه

اتصال‌های وب‌سوکت به یک اتصال TCP مداوم متکی هستند که می‌تواند با مشکلات شبکه، راه‌اندازی مجدد سرور یا عوامل دیگر قطع شود. پیاده‌سازی منطق اتصال مجدد در سمت کلاینت برای حفظ تجربه کاربری پایدار ضروری است.

1

## مقیاس‌پذیری

برخلاف HTTP که هر درخواست کوتاه‌مدت است، اتصال‌های وب‌سوکت طولانی‌مدت هستند. برای برنامه‌هایی با تعداد زیادی کلاینت، باید از متوازن‌کننده‌های بار آگاه از وب‌سوکت استفاده کنید و مطمئن شوید که زیرساخت سرور شما می‌تواند هزاران یا حتی میلیون‌ها اتصال همزمان را مدیریت کند.

4

## مدیریت منابع

اتصال‌های وب‌سوکت منابع سرور مانند حافظه و زمان پردازندۀ را مصرف می‌کنند. اگر اتصال‌ها به درستی مدیریت نشوند، می‌تواند منجر به اتمام منابع شود. استراتژی‌هایی برای بستن اتصال‌های غیرفعال پیاده‌سازی کنید.

3

## امنیت

اطمینان حاصل کنید که اتصال‌های وب‌سوکت شما روی یک کانال امن (wss://) به جای یک کانال رمزگذاری نشده (ws://) اجرا می‌شوند. اجرا روی TLS داده‌ها را رمزگذاری می‌کند و از شنود و حملات man-in-the-middle محافظت می‌کند.

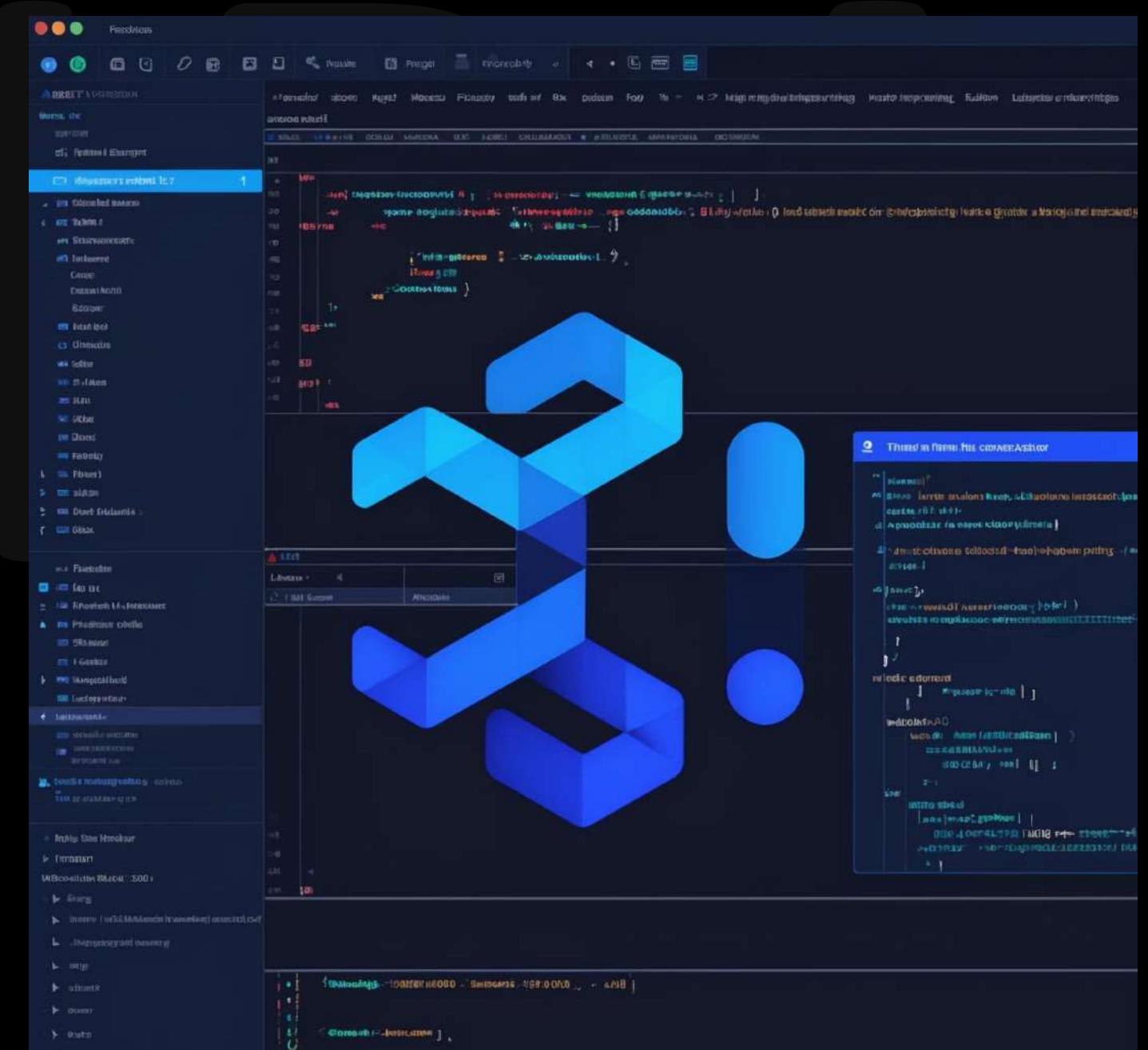
# معرفی وپسوکتها در سی‌شارپ

دات نت یک چارچوب قوی برای استفاده از وب سوکت‌ها ارائه می‌دهد که راه اندازی اجزای سمت سرور و سمت کلاینت را برای ارتباط بلا درنگ نسبتًا ساده می‌کند.

سمت سرور

این به شما امکان می‌دهد به راحتی .میان افزار یکپارچه‌ای برای مدیریت اتصال‌های وب‌سوکت ارائه می‌دهد ASP.NET Core ورودی را به یک اتصال وب‌سوکت ارتقا دهید، کانال پایدار را مدیریت کنید و تبادل پیام را در ساختار کد HTTP یک درخواست سی‌شاریپ معمول خود شروع کنید.

فضای نام System.Net.WebSockets تمام ابزارهای مورد نیاز برای شروع اتصال‌ها و ارسال یا دریافت پیام‌ها برای ارتباط وب‌سکت سمت کلاینت را فراهم می‌کند.



# راهاندازی وب‌سوکت‌ها در سی‌شارپ

راهاندازی سرور وب‌سوکت در ASP.NET Core

```
var builder = WebApplication.CreateBuilder(args); var app = builder.Build(); // فعالسازی پشتیبانی از وب‌سوکت app.UseWebSockets(); app.Use(async (context, next) => { if (context.Request.Path == "/ws") { if (context.WebSockets.IsWebSocketRequest) { var webSocket = await context.WebSockets.AcceptWebSocketAsync(); await EchoMessages(webSocket); } else { context.Response.StatusCode = 400; } } else { await next(); } }); app.Run(); // بک‌آپ تابع ساده برای پاسخ به پیام‌ها async Task EchoMessages(WebSocket webSocket){ var buffer = new byte[1024 * 4]; WebSocketReceiveResult result = await webSocket.ReceiveAsync( new ArraySegment(buffer), CancellationToken.None); while (!result.CloseStatus.HasValue) { await webSocket.SendAsync( new ArraySegment(buffer, 0, result.Count), result.MessageType, result.EndOfMessage, CancellationToken.None); result = await webSocket.ReceiveAsync( new ArraySegment(buffer), CancellationToken.None); } await webSocket.CloseAsync( result.CloseStatus.Value, result.CloseStatusDescription, CancellationToken.None); }
```

# راهاندازی و بسوکت‌ها در سی‌شارپ (ادامه)

ایجاد یک کلاینت و بسوکت ساده

```
به سرور "ws://localhost:5000/ws" متصل شد
سلام از طرف کلاینت"
پیام به سرور ارسال شد
پیام دریافت شده از سرور
در حال بستن
اتصال بسته شد"
```

using System.Net.WebSockets;using System.Text;using var client = new ClientWebSocket();await client.ConnectAsync(new Uri("ws://localhost:5000/ws"), CancellationToken.None);Console.WriteLine("به سرور \"ws://localhost:5000/ws\" متصل شد");var sendBuffer = Encoding.UTF8.GetBytes("سلام از طرف کلاینت");await client.SendAsync(new ArraySegment(sendBuffer), WebSocketMessageType.Text, true, CancellationToken.None);Console.WriteLine("پیام به سرور ارسال شد");var receiveBuffer = new byte[1024];var result = await client.ReceiveAsync(new ArraySegment(receiveBuffer), CancellationToken.None);Console.WriteLine(\$"پیام دریافت شده از سرور {Encoding.UTF8.GetString(result)}");await client.CloseAsync(WebSocketCloseStatus.NormalClosure, CancellationToken.None);Console.WriteLine("اتصال بسته شد");

htt

<https://github.com/MohamadHoseinRoohiAmini>

با این کد، ClientWebSocket به سرور در آدرس ws://localhost:5000/ws متصل می‌شود. کلاینت یک پیام به سرور ارسال می‌کند و سپس منتظر پاسخ می‌ماند.

# پیادهسازی یک سرور وب‌سوکت

برای پیادهسازی یک سرور وب‌سوکت که بتواند چندین اتصال را مدیریت کند، به مکانیزمی برای ردیابی هر کلاینت متصل و نمونه‌های وب‌سوکت آن‌ها نیاز داریم.

```
var clients = new Dictionary(); app.Use(async (context, next) =>{    if // دیکشنری برای ردیابی کلاینت‌های متصل /  
(context.Request.Path == "/ws") {        if (context.WebSockets.IsWebSocketRequest) {            var  
clientId = Guid.NewGuid().ToString();            var webSocket = await  
context.WebSockets.AcceptWebSocketAsync();            clients.Add(clientId, webSocket);  
Console.WriteLine($"کلاینت متصل شد: {clientId}");            await HandleClientCommunication(clientId,  
webSocket);        } else {            context.Response.StatusCode = 400;        }    } else {        await next();    }});
```

```
async Task HandleClientCommunication(string clientId, WebSocket webSocket){ var buffer = new byte[1024 * 4];  
try { WebSocketReceiveResult result = await webSocket.ReceiveAsync( new ArraySegment(buffer),  
CancellationToken.None); while (!result.CloseStatus.HasValue) { string receivedMessage =  
Encoding.UTF8.GetString( buffer, 0, result.Count); Console.WriteLine(${clientId}:  
{receivedMessage}); // ارسال پیام به همه کلاینت‌های متصل /  
foreach (var client in clients.Values) { if (client.State == WebSocketState.Open) { await client.SendAsync( new ArraySegment(buffer, 0, result.Count),  
result.MessageType, result.EndOfMessage, CancellationToken.None); } } result = await webSocket.ReceiveAsync(  
new ArraySegment(buffer), CancellationToken.None); } } catch (Exception ex) { Console.WriteLine($"خطا با  
کلاینت{clientId}: {ex.Message}"); } finally { clients.Remove(clientId); await webSocket.CloseAsync(  
WebSocketCloseStatus.NormalClosure, "اتصال بسته شد", CancellationToken.None); Console.WriteLine($"  
کلاینت{clientId} قطع شد"); }}
```

# پیاده‌سازی یک کلاینت وب‌سوکت

حال که یک سرور قادر به مدیریت چندین اتصال وب‌سوکت ایجاد کرده‌ایم، درک نحوه پیاده‌سازی یک کلاینت وب‌سوکت در سی‌شارپ ضروری است. این کلاینت یک مؤلفه کلیدی است که به ما امکان می‌دهد به سرور متصل شویم و با آن تعامل داشته باشیم.

```
using System.Net.WebSockets;using System.Text;using var client = new ClientWebSocket();try{    //    await client.ConnectAsync(new Uri("ws://localhost:5000/ws"),    CancellationToken.None);    Console.WriteLine("//    این کلاینت اتصال به سرور وب‌سوکت را بسته است.");    var sendBuffer = Encoding.UTF8.GetBytes("    سلام، سرور!");    await client.SendAsync(new ArraySegment(sendBuffer),    WebSocketMessageType.Text, true, CancellationToken.None);    Console.WriteLine("    پیام به سرور ارسال شد.");}
```

```
//    var receiveBuffer = new byte[1024];    while (client.State == WebSocketState.Open) {        var result = await client.ReceiveAsync(        new ArraySegment(receiveBuffer), CancellationToken.None);        if (result.MessageType == WebSocketMessageType.Close) {            Console.WriteLine("    سرور در حال بستن.");            await client.CloseAsync(            WebSocketCloseStatus.NormalClosure, CancellationToken.None);        } else {            string receivedMessage = Encoding.UTF8.GetString(            receiveBuffer, 0, result.Count);            Console.WriteLine($"    پیام دریافت شده از سرور: {receivedMessage}");        }    }    catch (Exception ex){        Console.WriteLine($"    : استثنای {ex.Message}");    }}
```

<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

این کلاینت به سرور وب‌سوکت متصل می‌شود، یک پیام سلام ارسال می‌کند و سپس وارد یک حلقه برای گوش دادن به پیام‌های سرور می‌شود.

# اشکال‌زدایی و تست



## ابزارهای تست وب‌سوکت

ابزارهای تست وب‌سوکت مانند (wscat) یک کلاینت وب‌سوکت خط فرمان (یا بهترین دوستان یک توسعه‌دهنده برای تست اولیه هستند. این Postman ابزارها فرآیند را ساده می‌کنند و به شما امکان می‌دهند به سرعت به سرور وب‌سوکت خود متصل شوید، پیام‌های تست ارسال کنید و پاسخ‌ها را مشاهده کنید.



## استفاده از لاغ‌گذاری دقیق

لاغ‌گذاری رویدادهای کلیدی مانند درخواست‌های اتصال، پیام‌های دریافتی و وضعیت قطع اتصال یکی از مؤثرترین روش‌ها برای اشکال‌زدایی وب‌سوکت‌ها است. این کار به شما کمک می‌کند تا آنچه در طول چرخه عمر یک اتصال اتفاق می‌افتد را ردیابی کنید.



## تست‌های واحد و یکپارچگی

تست‌های واحد به شما امکان می‌دهند اجزای خاصی را شبیه‌سازی کنید و منطق مدیریت پیام، سریالی‌سازی یا منطق داخلی سرور را آزمایش کنید. تست‌های یکپارچگی برای وب‌سوکت‌ها عملی‌تر هستند زیرا می‌توانند یک اتصال برقرار کنند، پیام‌ها را ارسال کنند و پاسخ را تأیید کنند.



## تست استرس

تست با چندین کلاینت که به طور همزمان متصل می‌شوند می‌تواند تصویر واضحی از نحوه مدیریت اتصال‌های همزمان توسط سرور شما ارائه دهد. تست استرس با حجم بالای پیام می‌تواند گلوگاه‌های عملکرد یا محدودیت‌ها در پیاده‌سازی سرور شما را آشکار کند.

<https://www.linkedin.com/in/mnramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# ویژگی‌های پیشرفته وب‌سوکت

## مدیریت گروههای کلاینت

برای برنامه‌های چت، ممکن است بخواهید "اتاق‌هایی" ایجاد کنید که در آن پیام‌ها فقط به گروه خاصی از کاربران پخش می‌شوند. برای دستیابی به این هدف، می‌توانید یک دیکشنری از گروههای کلاینت نگهداری کنید که شامل لیستی از اتصال‌های وب‌سوکت است.

## مدیریت انواع پیام

وب‌سوکت‌ها می‌توانند انواع مختلف پیام را مدیریت کنند: متنی، بازیزی و فریم‌های کنترلی. برای مثال، ممکن است بخواهید داده‌های بازیزی مانند تصاویر یا اشیاء سریالی‌شده را بین کلاینت‌ها ارسال کنید. این کار مستلزم شناسایی نوع پیام و اقدام مناسب با آن است.

## مدیریت اتصال‌های غیرفعال

اتصال‌های وب‌سوکت ماهیتاً پایدار هستند و باز نگه داشتن اتصال‌های استفاده نشده می‌تواند منجر به اتمام منابع شود. می‌توانید از پیام‌های دوره‌ای ping-pong برای بررسی فعال بودن کلاینت استفاده کنید و در صورت عدم فعالیت، اتصال را بیندید تا منابع سرور آزاد شوند.

## فشرده‌سازی وب‌سوکت

فشرده‌سازی می‌تواند کارایی انتقال داده را بهبود بخشد، به ویژه هنگام کار با محموله‌های بزرگ WebSocketDeflateOptions در ASP.NET Core. به شما امکان می‌دهد فشرده‌سازی پیام را فعال کنید، که گامی مهم در کاهش اندازه پیام‌های مبادله شده بین کلاینت و سرور است.

این ویژگی‌های پیشرفته به شما کنترل بیشتری بر نحوه تعامل سرور و کلاینت‌های وب‌سوکت می‌دهند و به شما امکان می‌دهند برنامه‌های بلادرنگ

پیچیده‌تری ایجاد کنید که کارآمد، مقیاس‌پذیر و قادر به مدیریت سناریوهای ارتباطی متنوع هستند:

<https://www.linkedin.com/in/mhrami/>  
<https://github.com/MohamadHoseinRoohiAmini>

# کار با WebRTC

فناوری ارتباطات بلادرنگ وب برای برنامه‌های .NET

ini/  
RoohiAmini



# فهرست مطالب

## معماری WebRTC

بررسی اجزای اصلی و نحوه عملکرد آنها

## ویژگی‌های کلیدی

ارتباط نقطه به نقطه، پشتیبانی از رسانه و داده، امنیت داخلی و سازگاری با شرایط شبکه

## مقدمه به WebRTC

آشنایی با فناوری ارتباطات بلادرنگ وب و مزایای آن

## موارد استفاده و چالش‌ها

کاربردهای عملی و مشکلات رایج در پیاده‌سازی

## راهاندازی اتصال نقطه به نقطه

سیگنالینگ و برقراری جلسه، عبور از شبکه با ICE، STUN و TURN

# فهرست مطالب (ادامه)

**کانال‌های داده و تبادل داده سفارشی**  
استفاده از کانال‌های داده برای ارتباطات بلادرنگ

**مدیریت جریان‌های رسانه‌ای**  
کار با جریان‌های صوتی و تصویری در برنامه‌های .NET.

**ادغام WebRTC در برنامه .NET.**  
پیاده‌سازی سرور سیگنالینگ و استفاده از Blazor در JavaScript Interop

**اشکال‌زدایی و آزمون برنامه‌های WebRTC**  
استفاده از ابزارهای توسعه‌دهنده مرورگر و نوشتن آزمون‌های واحد و یکپارچگی

**ملاحظات امنیتی**

ایمن‌سازی فرآیند سیگنالینگ و حفاظت از داده‌ها

# مقدمه به WebRTC

برنامه‌نویسی شبکه را با فراهم کردن امکان ارتباط نقطه به نقطه (ارتباطات بلادرنگ وب) بلادرنگ مستقیماً از مرورگرها یا برنامه‌های بومی متحول می‌کند.

این فناوری برای کنفرانس‌های ویدیویی، تماس‌های صوتی و اشتراک‌گذاری فوری داده بدون نیاز به افزونه‌ها یا تنظیمات پیچیده ایده‌آل است.

تأخیر کم و قابلیت‌های انتقال رسانه با کیفیت بالا، آن را برای ایجاد برنامه‌های بسیار تعاملی در اکوسیستم .NET 8 و C# مناسب می‌سازد.





# ارتباطات بلادرنگ را متحول می‌کند

با فعالسازی اتصالات مستقیم نقطه به نقطه برای تبادل WebRTC صوت، ویدیو و داده، ارتباطات بلادرنگ را متحول می‌کند و نیاز به سرورهای رسانه‌ای پیچیده را از بین می‌برد.

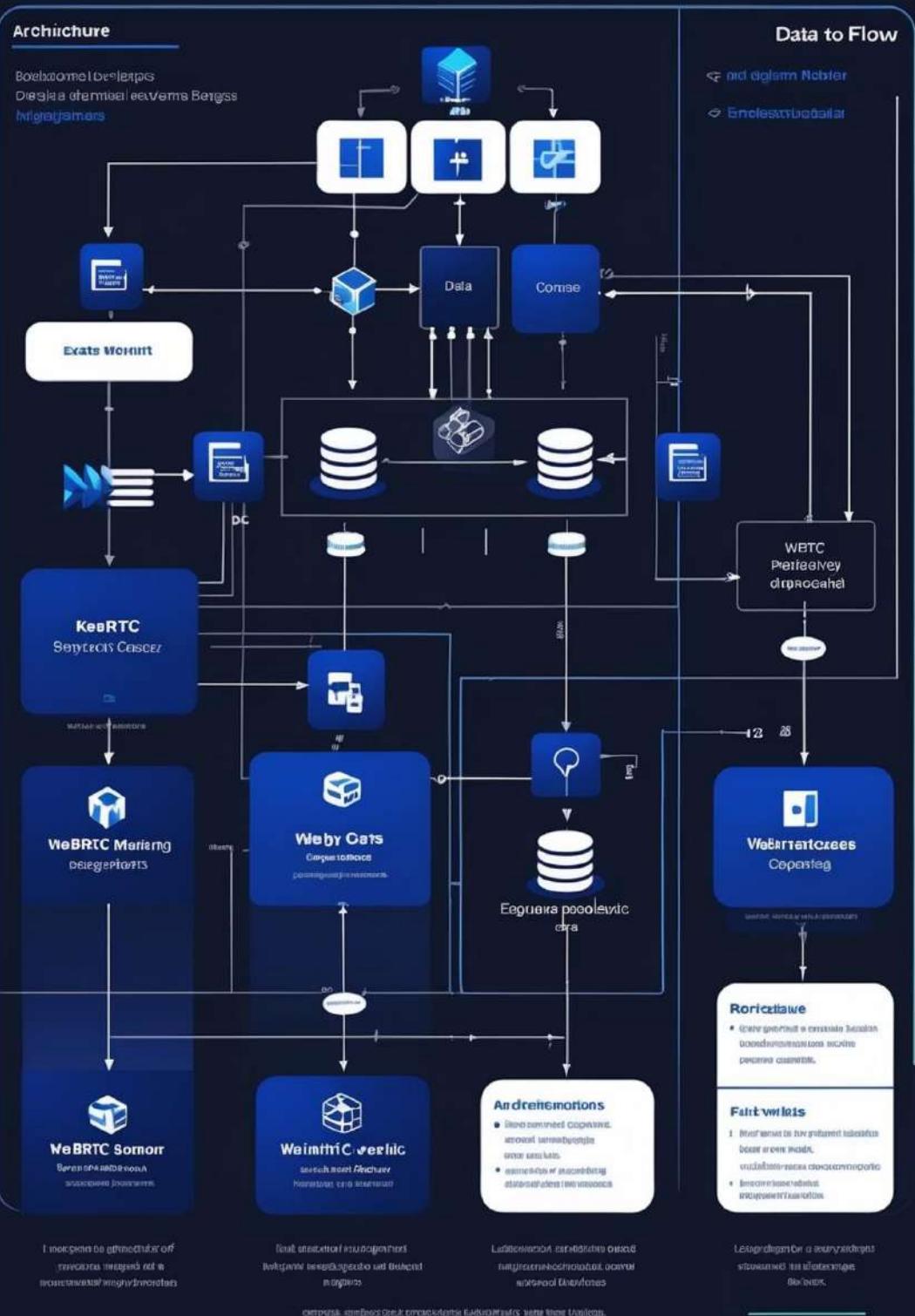
تعاملات با تأخیر کم و پهنانی باند بالای آن، این فناوری را برای چت‌های ویدیویی، ابزارهای همکاری و بازی‌های چندنفره ایده‌آل می‌سازد.

با بهره‌گیری از فناوری‌هایی مانند (STUN) ابزارهای عبور جلسه برای (NAT)، (TURN) عبور با استفاده از رله‌ها در اطراف (NAT و ICE) ابرقراری اتصال تعاملی ()، WebRTC اتصالات قابل اعتماد را در شبکه‌های مختلف یا فایروال‌ها تضمین می‌کند.

این رویکرد هزینه و پیچیدگی را در مقایسه با روش‌های سنتی وابسته به زیرساخت کاهش می‌دهد.

WEBRTC

Architectur



# نمای کلی معماری WebRTC

معماری WebRTC بر سه جزء اصلی استوار است که برای تسهیل ارتباط نقطه به نقطه بلادرنگ با هم کار می‌کنند: `RTCPeerConnection` و `MediaStream`. `RTCDataChannel` این اجزا توسط فرآیند سیگنالینگ پشتیبانی می‌شوند که متادیتای ضروری مانند پیام‌های پروتکل توصیف جلسه (SDP) و نامزدهای برقراری اتصال تعاملی (ICE) را برای برقراری اتصالات تبادل می‌کند.

# ویژگی‌های کلیدی WebRTC

به عنوان یک فناوری تحول‌آفرین برای ارتباطات بلادرنگ، مجموعه‌ای از ویژگی‌ها را برای برنامه‌های مدرن و تعاملی WebRTC ایده‌آل می‌سازد.

## پشتیبانی از رسانه و داده متنوع

مدیریت کارآمد جریان‌های رسانه‌ای (صوت و تصویر) و انتقال داده‌های دلخواه با تأخیر کم از طریق RTCDataChannel

## ارتباط نقطه به نقطه

برقراری اتصالات مستقیم بین دستگاه‌ها بدون نیاز به سرورهای مرکزی برای مسیریابی داده‌ها، کاهش تأخیر و هزینه‌های سرور

## سازگاری با شرایط شبکه

استفاده پویا از ICE، STUN و TURN برای حفظ ثبات اتصالات حتی در سناریوهای چالش‌برانگیز مانند تغییر بین شبکه‌های Wi-Fi و موبایل

## امنیت داخلی

رمزگذاری یکپارچه با استفاده از DTLS برای سیگنالینگ و RTP برای جریان‌های رسانه‌ای، تضمین حریم خصوصی و محافظت در برابر دستکاری

# اجزای اصلی معماری WebRTC

## RTCPeerConnection

مدیریت لجستیک اتصالات نقطه به نقطه،  
تضمين اتصال امن با استفاده از پروتکل های  
رمزگذاری و تنظیم پویا با شرایط شبکه

## فرآیند سیگنالینگ

تسهیل تبادل متادیتای مورد نیاز برای برقراری  
اتصال، از جمله پیام های SDP و نامزدهای  
ICE

## MediaStream

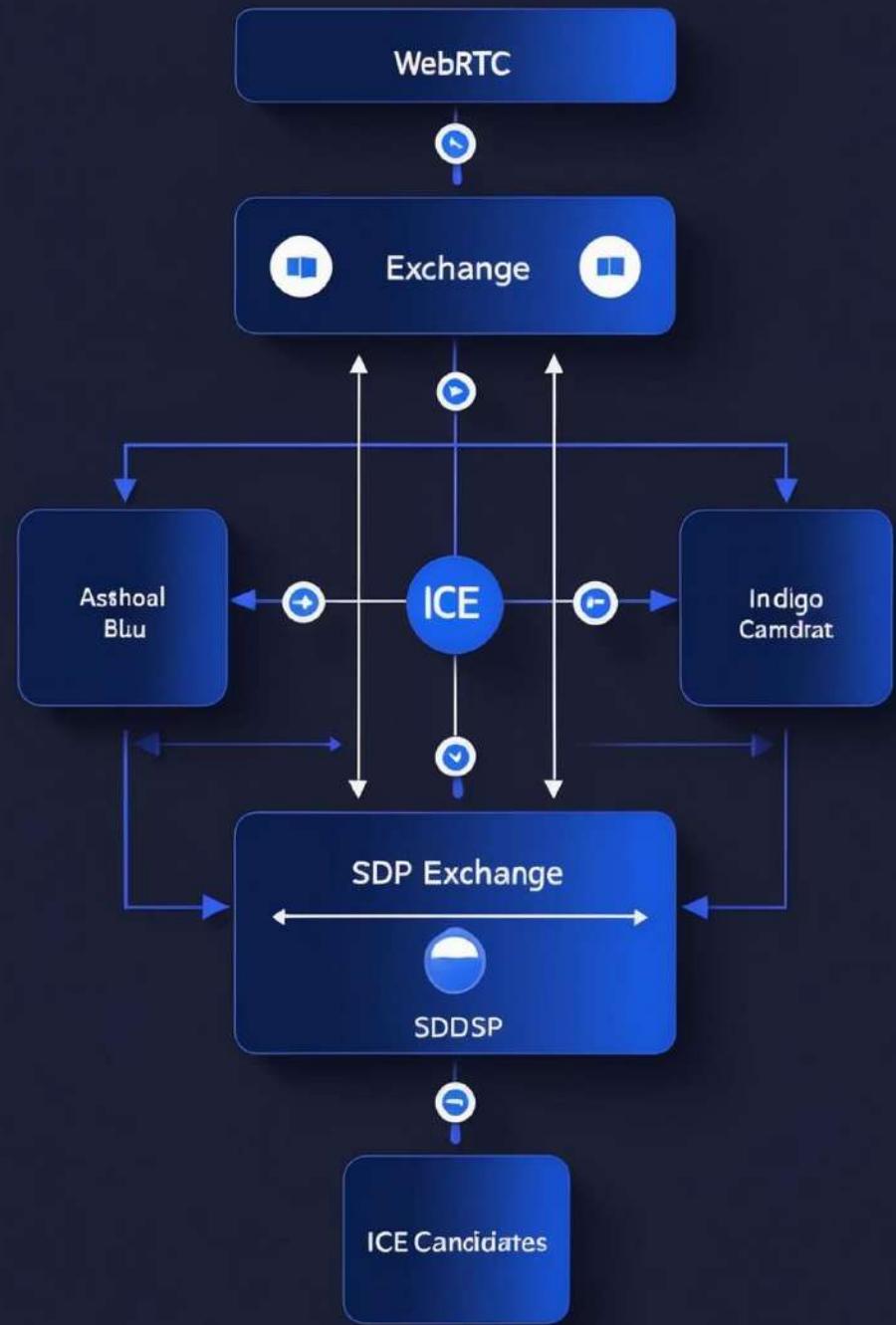
مدیریت جریان های صوتی و تصویری، انتزاع  
پیچیدگی ضبط، کدگذاری و انتقال محتوای  
چند رسانه ای

## RTCDataChannel

امکان انتقال داده های دلخواه با تأخیر کم و  
دو طرفه، مناسب برای اشتراک گذاری اطلاعات  
بازی، ویرایش متن مشارکتی یا اشتراک فایل



## WebRTC Signalling Signal



# نودار جریان سیگنالینگ

فرآیند سیگنالینگ به عنوان پلی عمل می‌کند که این اجزا را با تسهیل تبادل متادیتای مورد نیاز برای برقراری اتصال به هم متصل می‌کند. پیام‌های SDP جزئیاتی درباره قابلیت‌های رسانه‌ای، کدک‌ها و الزامات رمزگذاری ارائه می‌دهند و اطمینان حاصل می‌کنند که هر دو همتا در مورد شرایط ارتباط توافق دارند. نامزدهای ICE اکشاف مسیرهای بهینه شبکه را امکان‌پذیر می‌کنند و بر چالش‌های ناشی از فایروال‌ها و NAT‌ها غلبه می‌کنند.

# راهاندازی اتصال نقطه به نقطه WebRTC

برقراری اتصال نقطه به نقطه در WebRTC شامل مجموعه‌ای از مراحل هماهنگ است که سیگنالینگ، عبور از شبکه و مدیریت اتصال را با هم ترکیب می‌کند.



## برقراری اتصال

پس از تکمیل سیگنالینگ و عبور از شبکه،  
RTCPeerConnection مدیریت تبادل واقعی  
رسانه و داده را بر عهده می‌گیرد



## عبور از شبکه با ICE، STUN و TURN

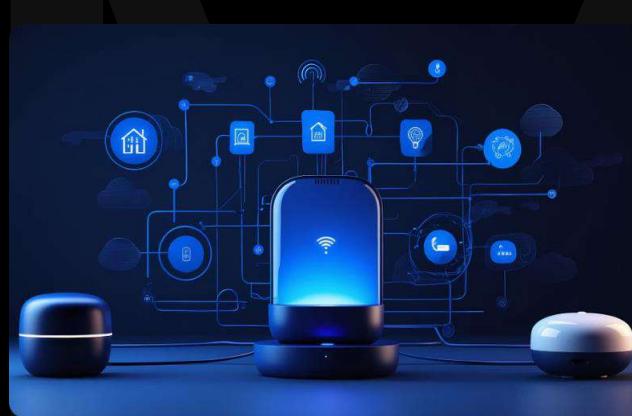
غلبه بر موانع شبکه مانند فایروال‌ها و NAT‌ها  
با استفاده از مکانیسم‌های ICE، ICE و STUN و  
TURN برای کشف و برقراری مسیرهای  
ارتباطی کارآمد



## سیگنالینگ و برقراری جلسه

تبادل متادیتای حیاتی مانند پیام‌های SDP و  
نامزدهای ICE بین همتایان برای مذاکره  
پارامترهای ارتباطی آنها

# موارد استفاده WebRTC



## اینترنت اشیاء (IoT)

دستگاه‌های هوشمند می‌توانند داده‌ها را مستقیماً به یکدیگر یا به سیستم‌های مرکزی به صورت بلاذرنگ منتقل کنند.



## پخش زنده

پلتفرم‌های میزبانی حراجی‌ها، گزارش‌های ورزشی یا رویدادهای زنده از WebRTC برای ارائه جریان‌های بلاذرنگ استفاده می‌کنند.



## بازی‌های آنلاین

کانال‌های داده با تأخیر کم به توسعه‌دهندگان بازی اجازه می‌دهد وضعیت بازی را بین بازیکنان به صورت بلاذرنگ همگام‌سازی کنند.



## کنفرانس ویدیویی

معماری نقطه به نقطه WebRTC تعاملات روان و با کیفیت بالا را تضمین می‌کند، که آن را برای تماس‌های یک به یک و جلسات مجازی بزرگ مناسب می‌سازد.

# چالش‌های WebRTC

## تضمين امنیت

در حالی که WebRTC رمزگذاری داخلی برای جریان‌های رسانه‌ای و داده فراهم می‌کند، پیاده‌سازی سیگنالینگ امن به عهده توسعه‌دهندگان است.

بدون محافظت‌های مناسب، مانند HTTPS و مکانیسم‌های احراز هویت قوی، کانال‌های سیگنالینگ می‌توانند در برابر حملاتی مانند شنود و جعل آسیب‌پذیر باشند.

مدیریت مجوزها برای دسترسی به منابع حساس، مانند دوربین‌ها و میکروفون‌ها، لایه دیگری از مسئولیت را اضافه می‌کند.

## عبور از شبکه

برقراری اتصال نقطه به نقطه بین دستگاه‌ها اغلب نیازمند عبور از موانعی مانند NAT‌ها و فایروال‌ها است. WebRTC از مکانیسم‌هایی مانند ICE، TURN و STUN برای غلبه بر این موانع استفاده می‌کند.

اگرچه این ابزارها امکان اتصال را در اکثر سناریوهای فراهم می‌کنند، اما پیچیدگی‌هایی را در زمان راه‌اندازی معرفی می‌کنند و نیاز به پیکربندی دقیق را افزایش می‌دهند.

# ادغام WebRTC در برنامه .NET

ادغام WebRTC در یک برنامه .NET، معماری، اجزا و تنظیم اتصال نقطه به نقطه را به واقعیت تبدیل می‌کند. این بخش بر پیاده‌سازی عملی تمرکز دارد و بررسی می‌کند که چگونه می‌توان یک بک‌اند .NET را با کلاینت‌های WebRTC برای ارتباط بلادرنگ و با تأخیر کم هماهنگ کرد.

## مدیریت سیگنالینگ

یک عنصر کلیدی این ادغام، مدیریت فرآیند سیگنالینگ است که شامل تبادل توصیف‌های جلسه و نامزدهای ICE بین همتایان است. برای تسهیل این امر، یک سرور سیگنالینگ در ASP.NET Core با استفاده از WebSocket‌ها راه‌اندازی می‌شود.

## تبادل رسانه و داده

فراتر از سیگنالینگ، ادغام باید از تبادل روان‌رسانه و داده پشتیبانی کند، جایی که .NET مذاکره رسانه، مدیریت جلسه و حفظ کیفیت اتصال را تسهیل می‌کند.

استفاده از JavaScript interop در برنامه‌های Blazor، بک‌اند .NET را با API سمت کلاینت WebRTC به طور یکپارچه پیوند می‌دهد.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# پیاده‌سازی سرور سیگنالینگ در .NET.

راه اندازی یک سرور سیگنالینگ بخش حیاتی از کارکرد WebRTC است و با .NET، شما تمام ابزارهای لازم برای انجام مؤثر آن را دارید. هدف سرور سیگنالینگ تسهیل تبادل توصیف‌های جلسه و نامزدهای ICE بین همتایان است.

```
var builder = WebApplication.CreateBuilder(args); var app =  
builder.Build(); app.UseWebSockets(); app.UseRouting(); app.UseEndpoints(endpoints => {  
    endpoints.MapGet("/signal", async context => {  
        await HandleWebSocketRequest(context);    });}); app.Run();
```

در کد بالا، ما یک نقطه پایانی /signal تعریف کرده‌ایم که به عنوان نقطه ورودی برای ارتباط WebSocket عمل می‌کند. این جایی است که کلاینت‌ها برای تبادل پیام‌های سیگنالینگ خود متصل می‌شوند.  
<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

# استفاده از WebRTC برای JavaScript Interop در Blazor

یکی از عملی ترین رویکردها برای ادغام این فناوری با Blazor، استفاده از JavaScript interop است. ابزاری فوق العاده برای ایجاد برنامه های وب مدرن در #C است، اما WebRTC عمدهاً قلمرو JavaScript است.

```
// wwwroot/webrtc.jslet localConnection = null;let remoteConnection =  
null;window.webRTCTInterop = {    initializeConnection: function () {  
localConnection = new RTCPeerConnection();        remoteConnection = new  
RTCPeerConnection();        // ICE مدیریت نامزدهای  
localConnection.onicecandidate = (event) => {            if (event.candidate) {  
remoteConnection.addIceCandidate(event.candidate);        }    };  
// دور مدیریت جریان از راه دورremoteConnection.ontrack = (event) => {  
const remoteVideo = document.getElementById("remoteVideo");  
remoteVideo.srcObject = event.streams[0];    };}};
```

```
@page "/"/@inject IJSRuntime JS
```

```
...ode { private async Task InitializeConnection() { await JS.InvokeVoidAsync("we@
```

چت تصویری WebRTC

ش...

...

# مدیریت جریان‌های رسانه‌ای در برنامه‌های .NET.

مدیریت جریان‌های رسانه‌ای یکی از قابلیت‌های اصلی است که WebRTC را قادرمند می‌سازد و امکان تبادل صوت و تصویر بladرنگ بین همتایان را فراهم می‌کند. در یک برنامه .NET، چالش در ادغام مؤثر فراتاند (جایی که کاربران با ویدیو و صدا تعامل دارند) (با بکاندی است که قادر به پشتیبانی از سیگنالینگ و مدیریت تبادل داده بladرنگ باشد).



ارتباط پیشنهادهای SDP، پاسخها و نامزدهای ICE ابه بکاند .NET. که سپس پیام‌ها را بین کلاینت‌ها رله می‌کند.

استفاده از ()getUserMedia() برای ضبط جریان‌های صوتی و تصویری از دستگاه محلی و افزودن آن به اتصال همتا.

## نظارت و بهینه‌سازی

حفظ اتصال، نظارت بر کیفیت و تنظیم پارامترهایی مانند نرخ بیت،  
وضوح یا روشن/خاموش کردن ترک‌های خاص.

## نمایش و کنترل رسانه

نمایش رسانه در عناصر ویدیویی و کنترل جریان از طریق JavaScript interop، با امکان افزودن یا حذف پویایی ترک‌ها.

# کانال‌های داده و تبادل داده سفارشی

کانال‌های داده منبع الهام بی‌پایانی در چشم‌انداز WebRTC هستند. آنها از پخش صوت و تصویر معمولی فراتر می‌روند و دنیایی را باز می‌کنند که در آن پیام‌های بلادرنگ، اشتراک‌گذاری فایل و همگام‌سازی وضعیت برنامه می‌تواند مستقیماً بین کاربران، با حداقل تأخیر رخ دهد.

## مدیریت کانال‌های داده WebRTC

کانال‌های داده، ویژگی قدرتمند WebRTC، امکان انتقال مستقیم داده‌های دلخواه بین همتایان را فراهم می‌کنند. برخلاف جریان‌های رسانه‌ای که برای صوت و تصویر استفاده می‌شوند، کانال‌های داده به شما امکان می‌دهند متن، فایل‌ها، وضعیت‌های بازی یا هر داده سفارشی را با تأخیر کم ارسال کنید.

```
// wwwroot/webrtc.jslet dataChannel = null;window.webRTCTInterop = {  
    initializeConnection: function () { // کانال داده dataChannel =  
        localConnection.createDataChannel("chat"); dataChannel.onopen = () =>  
            console.log("کانال داده باز است"); dataChannel.onmessage = (event) =>  
                console.log("پیام دریافت شد:", event.data); // نمایش پیام دریافتی در  
                sendMessage: function (message) { if (dataChannel &&  
                    dataChannel.readyState === "open") { dataChannel.send(message);  
                    console.log("پیام ارسال شد:", message); } }};
```

## تنظیم نرخ بیت و وضوح

یکی از عوامل مهم در حفظ تجربه رسانه‌ای با کیفیت بالا در WebRTC، توانایی تنظیم پویای نرخ بیت و وضوح جریان‌های ویدیویی است. شرایط شبکه می‌تواند بین کاربران به طور قابل توجهی متفاوت باشد و بدون تنظیمات مناسب، ممکن است با اتصالی مواجه شوید که یا کیفیت پایینی دارد یا پهنای باند زیادی مصرف می‌کند.

```
// wwwroot/webrtc.jswindow.webRTCInterop = {    adjustBitrate:  
function (bitrate) {        const senders =  
localConnection.getSenders();        senders.forEach(sender => {  
if (sender.track && sender.track.kind === 'video') {  
const parameters = sender.getParameters();                if  
(!parameters.encodings) {                    parameters.encodings =  
[{}];                }  
parameters.encodings[0].maxBitrate = bitrate * 1000;  
sender.setParameters(parameters);                console.log(`نرخ  
ビットウェイブレット ${bitrate} kbps でビデオを設定しました`);  
}};
```

```
@page "/"
```

## تنظیم نرخ بیت ویدیوی WebRTC

```
...ode {    private int bitrate;    private async Task SetBitrate()  {      if@     ...} }
```

<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

# ملاحظات امنیتی در WebRTC

در زمینه ارتباطات بلادرنگ، امنیت نه یک فکر ثانویه، بلکه یک نیاز اساسی است. در WebRTC، ملاحظات امنیتی از ابتدا در پروتکل گنجانده شده‌اند، تضمین می‌کند که تبادل صوت، تصویر و داده بین همتایان خصوصی و امن باشد.

## کنترل دسترسی به منابع

ملاحظه مهم دیگر در امنیت WebRTC، کنترل دسترسی به منابع ارتباطی مانند دوربین‌ها، میکروفون‌ها و کانال‌های داده مشترک است. کاربران باید اطمینان داشته باشند که به دستگاه‌های آنها به طور مناسب دسترسی پیدا می‌شود و داده‌های حساس فقط با همتایان مجاز به اشتراک گذاشته می‌شود.

## ایمن‌سازی کانال‌های سیگنالینگ

آسیب‌پذیری سایر جنبه‌ها، مانند سرور سیگنالینگ، می‌تواند توسط مهاجمان برای ریودن جلسه مورد سوءاستفاده قرار گیرد. بنابراین، ایمن‌سازی کانال‌های سیگنالینگ، اغلب از طریق HTTPS و WebSocket‌ها با احراز هویت مناسب، جنبه مهمی از امنیت WebRTC است.

## رمزگذاری داخلی

تمام جریان‌های رسانه‌ای و WebRTC داده را به طور پیش‌فرض رمزگذاری برای ایمن‌سازی DTLS می‌کند، از برای ایمن‌سازی RTP کانال‌های داده و با این حال، امنیت رسانه استفاده می‌کند برنامه شما تنها به رمزگذاری وابسته نیست.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# اشکال‌زدایی و آزمون برنامه‌های WebRTC

## استفاده از ابزارهای توسعه‌دهنده مرورگر

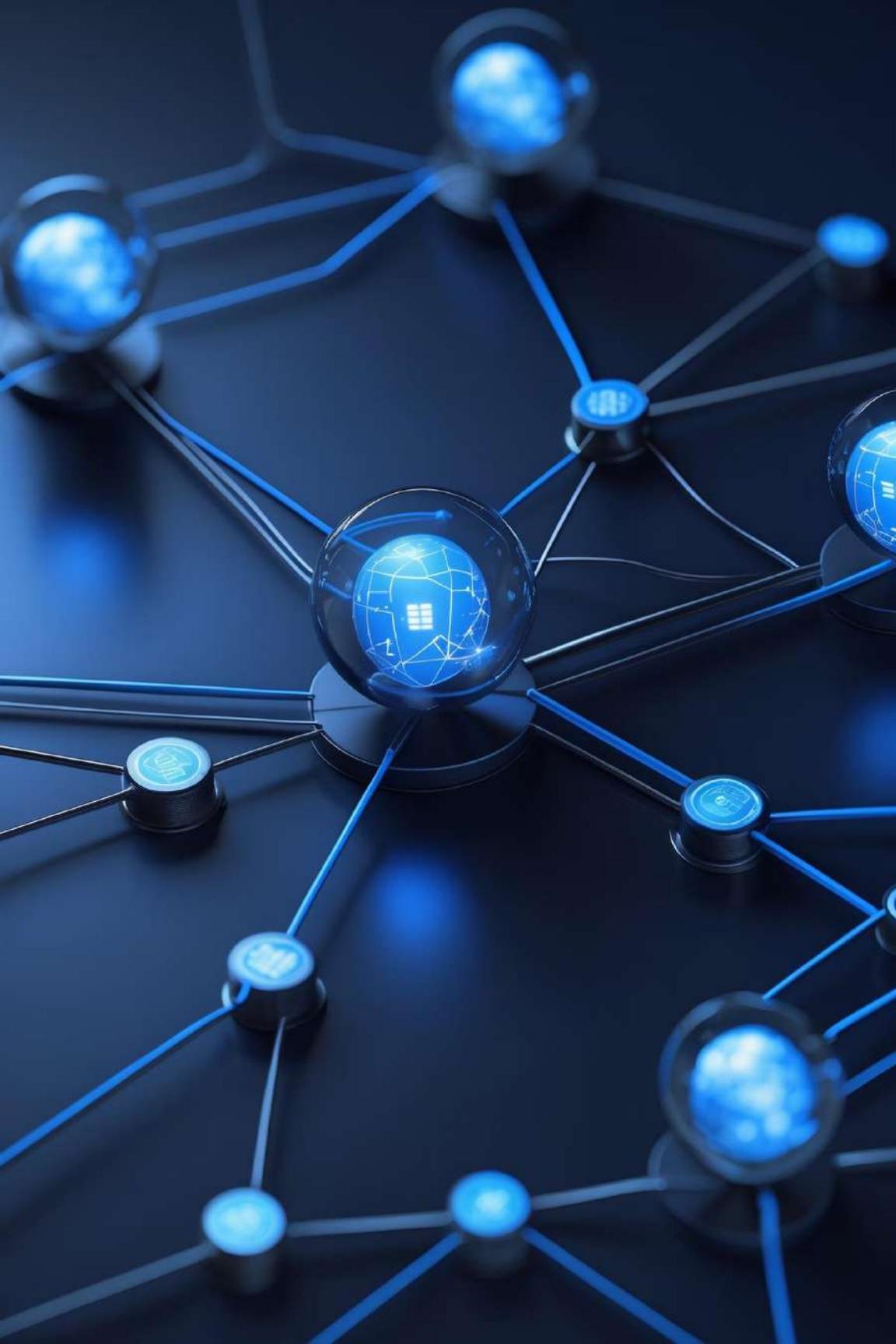
ابزارهای توسعه‌دهنده مرورگر نقشی اساسی در بهینه‌سازی عملکرد برنامه فناوری، با نوشتن آزمون‌های واحد برای برنامه‌های سنتی کلاینت-سرور متفاوت است. با این حال، یک استراتژی خوب این است که آزمون را به دو بخش تقسیم کنید: آزمون‌های واحد برای اجزای مجزا مانند منطق سیگنالینگ و آزمون‌های یکپارچگی که کل جریان سیگنالینگ و تبادل رسانه را تأیید می‌کنند.

یکی از ارزشمندترین ویژگی‌ها در Chrome، صفحه-`chrome://webrtc-internals` است که گزارش مفصلی از تمام جلسات فعل WebRTC در مرورگر ارائه می‌دهد. این ابزار به شما امکان می‌دهد تبادلات SDP را نظارت کنید، نامزدهای ICE را پیگیری کنید و آمار بلادرنگ مانند نرخ بیت، از دست دادن بسته و لرزش را مشاهده کنید.

## نوشتن آزمون‌های واحد و یکپارچگی

آزمون منطق WebRTC به دلیل ماهیت بلادرنگ و نقطه به نقطه این فناوری، با نوشتن آزمون‌های واحد برای برنامه‌های سنتی کلاینت-سرور متفاوت است. با این حال، یک استراتژی خوب این است که آزمون را به دو بخش تقسیم کنید: آزمون‌های واحد برای اجزای مجزا مانند منطق سیگنالینگ و آزمون‌های یکپارچگی که کل جریان سیگنالینگ و تبادل رسانه را تأیید می‌کنند.

آزمون‌های واحد برای موارد حاشیه‌ای، ورودی‌های نادرست و حالت‌های غیرمنتظره که می‌توانند باعث مشکلات سیگنالینگ شوند، عالی هستند.



# کار با gRPC

یک چارچوب قدرتمند برای ارتباطات سریع، کارآمد و مقیاس‌پذیر بین سرویس‌ها

ini/  
RoohiAmini

# فهرست مطالب

## معماری و پروتکل‌ها

درک معماری gRPC و پروتکل‌های زیربنایی

## مقدمه و مفاهیم اصلی

آشنایی با gRPC و نقش آن در برنامه‌های مدرن

## ویژگی‌های پیشرفته

استریمینگ، امنیت و بهینه‌سازی عملکرد

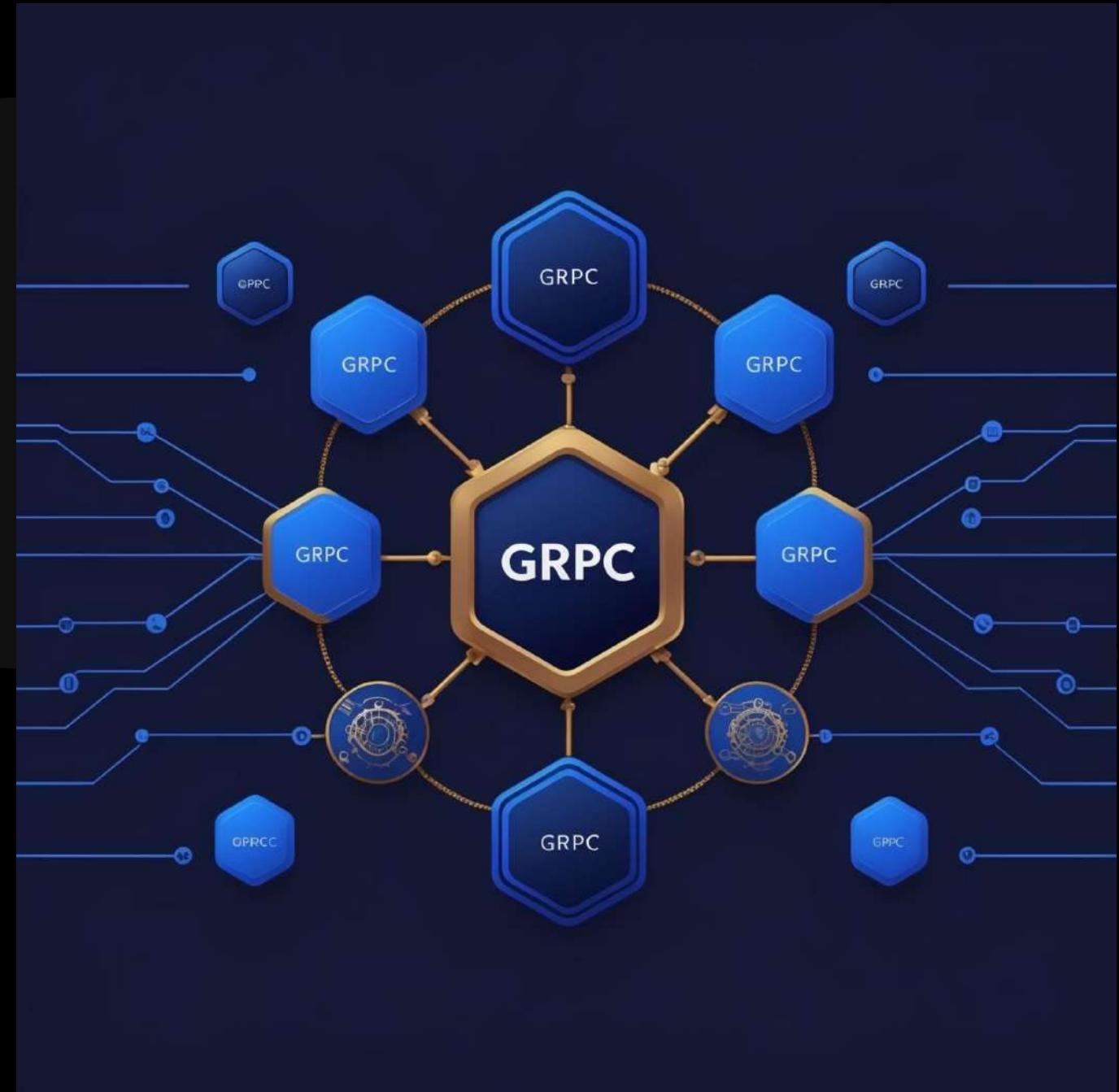
## راهاندازی سرویس‌ها

ایجاد و پیکربندی سرویس‌های gRPC در .NET

# معرفی gRPC و نقش آن در برنامه‌های مدرن

یک چارچوب قدرتمند است که به عنوان ابزاری کارآمد برای ساخت ارتباطات سریع، کارآمد و سنتی پیشی می‌گیرد، REST API‌های کارایی آن، که از مقیاس‌پذیر بین سرویس‌ها ظهر کرده است است که ویژگی‌هایی مانند Protocol Buffers (Protobuf) و HTTP/2 به دلیل استفاده از استریمینگ دوطرفه، مالتی‌پلکسینگ و سریالیزه کردن فشرده پیام‌ها را امکان‌پذیر می‌کند.

با .NET 8، gRPC به یک شهروند درجه یک در اکوسیستم .NET تبدیل شده است و پشتیبانی قوی برای ایجاد سرویس‌ها و کلاینت‌های gRPC ارائه می‌دهد. ادغام بی‌نقص آن با C# اطمینان از یک تجربه توسعه مطمئن و قابل اعتماد را فراهم می‌کند، با کلاس‌های تولید شده و API‌های شهودی که بسیاری از کارهای سنگین را برای شما انجام می‌دهند.



<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

# مقایسه با REST و سایر پروتکل‌ها



## REST

- استفاده از HTTP و معمولاً JSON برای تبادل داده
- ساده و گسترده
- سریالیزه کردن متنی ناکارآمد
- فاقد استریمینگ بومی



## gRPC

- استفاده از Protobuf و HTTP/2
- سریالیزه کردن با اینری فشرده و سریع
- پشتیبانی از استریمینگ دوطرفه
- قراردادهای تایپ شده قوی



## سایر پروتکل‌ها

- محور-برای ارتباطات رویداد
- برای پرس و جوهای انعطاف‌پذیر
- هر کدام در حوزه‌های خاص خود برتری دارد

# موارد استفاده رایج برای gRPC

## برنامه‌های موبایل و محاسبات لبه

با اندازه پیام‌های کاهش یافته و توانایی کار در شبکه‌های محدود، gRPC برای برنامه‌های موبایل که با سرویس‌های بکاند ارتباط برقرار می‌کنند یا دستگاه‌های لبه که داده‌ها را با سیستم‌های مرکزی تبادل می‌کنند، مناسب است.

## استریمینگ داده‌های بلاذرنگ

برنامه‌هایی مانند بهروزرسانی‌های ورزشی زنده، فیدهای بازار مالی و تله‌متری IoT ایاز به تبادل داده مداوم و دوطرفه بین کلاینت‌ها و سورورها دارند. پشتیبانی gRPC از RPC‌های استریمینگ، پیاده‌سازی بی‌نقص این سناریوها را تضمین می‌کند.

## معماری میکروسرویس

برای ارتباط سرویس به سرویس gRPC ایده‌آل است، جایی که سرویس‌ها باید به طور مکرر و سریع داده‌ها را تبادل کنند. فرمت سریالیزه کردن فشرده و ویژگی‌های باعث کاهش سربار و بهبود توان HTTP/2 عملیاتی می‌شود.

# چگونه gRPC در معماری‌های برنامه مدرن جای می‌گیرد

این فناوری به طور مؤثر یک راه حل قدرتمند است که نقش محوری در معماری‌های برنامه مدرن ایفا می‌کند gRPC چالش‌های ارتباط کارآمد و قابل اعتماد در سیستم‌های توزیع شده، به ویژه در معماری‌های مبتنی بر میکروسرویس را برطرف می‌کند.

فراتر از میکروسرویس‌ها، gRPC به طور یکپارچه در اکوسیستم‌های بومی ابر ادغام می‌شود. کشف سرویس آن، که به سرویس‌ها اجازه می‌دهد یکدیگر را بدون کدگذاری سخت مکان‌هایشان پیدا کنند و با هم ارتباط برقرار کنند، و پشتیبانی از متعادل‌سازی باشد، که ترافیک شبکه ورودی را در میان گروهی از سوررهای بکاند توزیع می‌کند، به خوبی با پلتفرم‌های ارکستراسیون کانتینر مانند Kubernetes هماهنگ است.



<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

# درک معماری gRPC و پروتکل‌ها

برای بهره‌برداری کامل از قدرت gRPC، درک معماری و پروتکل‌های زیربنایی آن ضروری است. در هسته خود، gRPC یک چارچوب ارتباطات با کارایی بالا طراحی شده است. این چارچوب بر اساس یک مدل کلاینت-سرور عمل می‌کند، جایی که کلاینت‌ها متدهایی را روی سرورهای راه دور فراخوانی می‌کنند، گویی که محلی هستند. این انتزاع از طریق تعاریف سرویس با تایپ قوی، که توسط Protobuf فعال شده است، به دست می‌آید.

معماری gRPC به شدت با HTTP/2، یک پروتکل انتقال مدرن با ویژگی‌های مالتی‌پلکسینگ، استریمینگ دوطرفه و فشرده‌سازی هدر، مرتبط است. برخلاف ماهیت بدون حالت HTTP/1.1 است، HTTP/2 اجازه اتصالات پایدار را می‌دهد، که به چندین جریان امکان می‌دهد بدون سربار باز کردن اتصالات جدید، به طور همزمان عمل کنند.

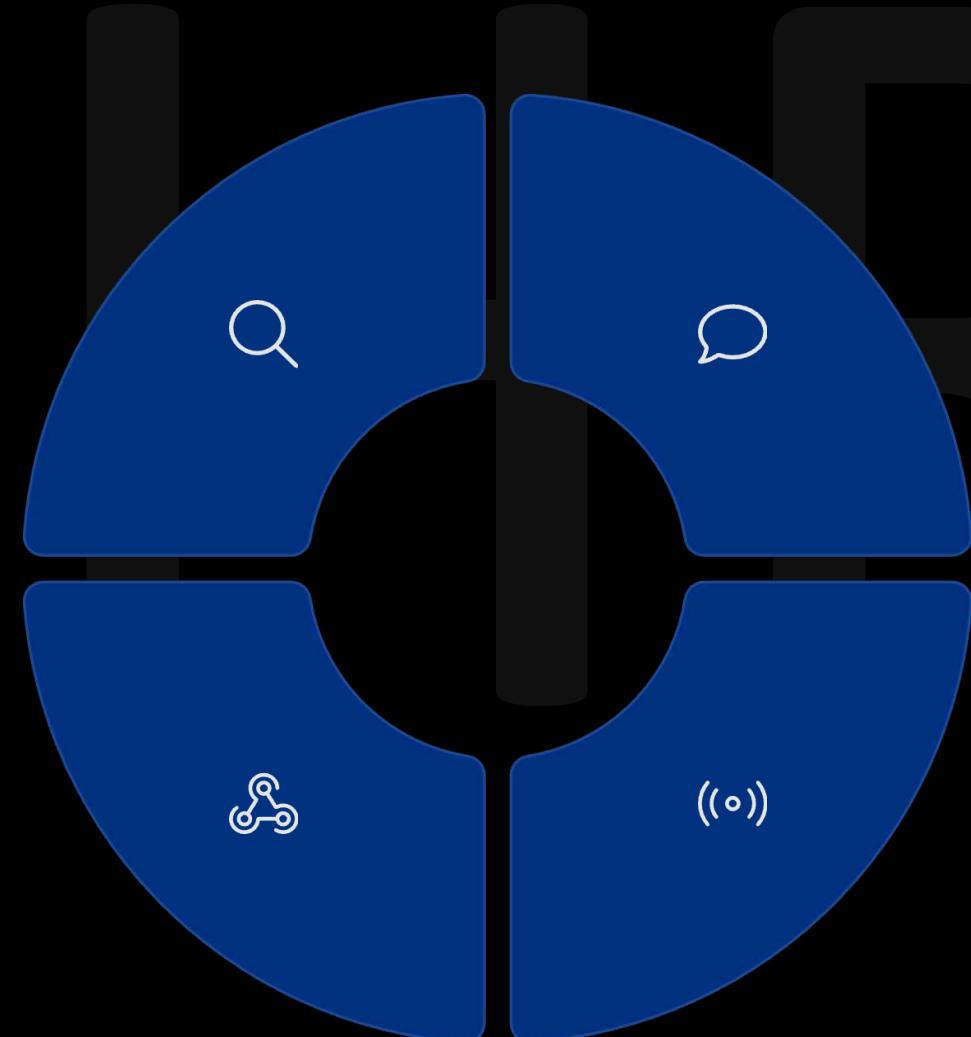
# مفاهیم اصلی gRPC

## Protobuf

زبان تعریف رابط و مکانیسم سریالیزه کردن پیام. به توسعه دهنده‌گان اجازه می‌دهد قراردادهای سرویس و ساختارهای پیام را در یک فایل .proto تعریف کنند.

## HTTP/2

پروتکلی که ویژگی‌های انتقال پیشرفته ضروری برای برنامه‌های مدرن را فراهم می‌کند. مالتی‌پلکسینگ چندین جریان روی یک اتصال TCP واحد را فعال می‌کند.



## پارادایم RPC

کلاینت می‌تواند مستقیماً متدهایی را روی سرور فراخوانی کند، گویی که یک شیء محلی است، پیچیدگی‌های ارتباط شبکه زیربنایی را پنهان می‌کند.

## الگوهای ارتباطی

پشتیبانی از چهار نوع RPC: یکتایی، استریمینگ سرور، استریمینگ کلاینت و استریمینگ دوطرفه، که به gRPC اجازه می‌دهد با نیازهای مختلف برنامه سازگار شود.

# قابلیت گسترش و همکاری

یکی از بزرگترین نقاط قوت gRPC در قابلیت گسترش و همکاری آن نهفته است، که آن را به یک چارچوب چند منظوره برای ساخت سیستم‌های توزیع شده تبدیل می‌کند. در هسته خود، gRPC طراحی شده است تا بدون مشکل در چندین زبان برنامه‌نویسی و پلتفرم کار کند، ارتباط کارآمد بین اجزای مختلف یک سیستم را تضمین می‌کند، صرف نظر از پیاده‌سازی زیربنایی آنها.

قابلیت گسترش در gRPC از طریق ویژگی‌هایی مانند ایترسپیتورها و متادیتای سفارشی به دست می‌آید. ایترسپیتورها به توسعه‌دهندگان اجازه می‌دهند نگرانی‌های فرآگیر مانند لاغینگ، نظارت و احراز هویت را بدون تغییر منطق اصلی سرویس پیاده‌سازی کنند.



<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

# راه اندازی یک سرویس gRPC در .NET

## تعریف قرارداد سرویس با Protobuf

فایل .proto را در پوشه Protos ویرایش کنید تا سرویس و پیام‌های خود را تعریف کنید.

## ایجاد پروژه gRPC

با استفاده از CLI دات‌نت یک پروژه جدید ایجاد کنید:

```
dotnet new grpc -n GrpcExample
```

## پیکربندی و اجرای سرویس

dotnet run کنید و با gRPC را برای پیکربندی سرویس Program.cs تنظیم کنید. اجرا کنید.

## پیاده‌سازی منطق سرویس

کلاس پایه سرویس تولید شده را در Services/ گسترش دهید تا منطق کسب و کار خود را پیاده‌سازی کنید.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

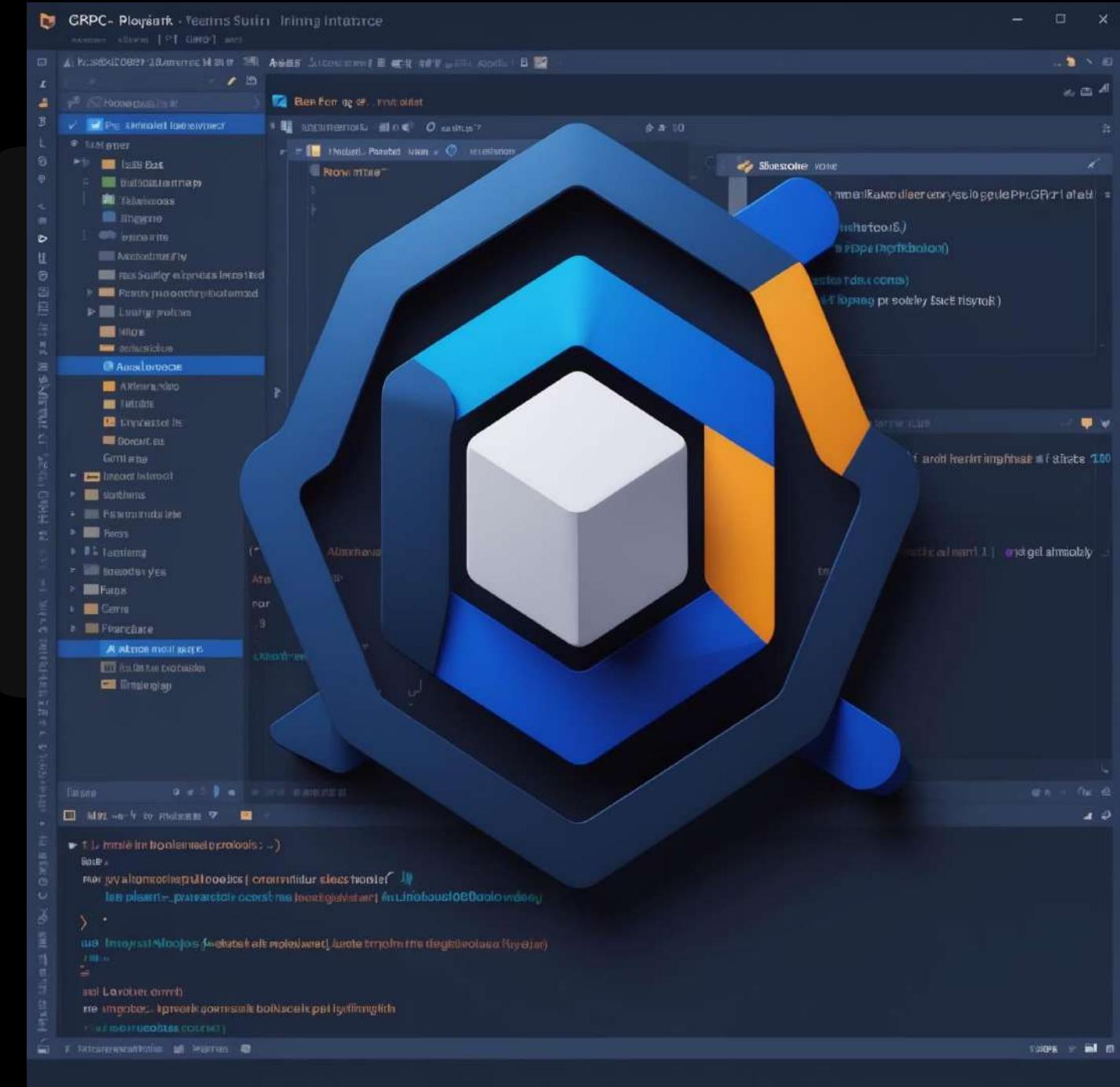
# ایجاد یک پروژه gRPC در .NET

ایجاد یک پروژه gRPC در .NET ساده است، به لطف ابزارها و قالب‌های قدرتمند ارائه شده توسط چارچوب چه از CLI داتننت استفاده کنید، فرآیند طراحی شده است تا شما را سریعاً با یک سرویس gRPC راهاندازی کند.

با ایجاد یک پروژه gRPC جدید با استفاده از CLI داتننت شروع کنید:

```
dotnet new grpc -n GrpcExample
```

این دستور یک پروژه gRPC جدید به نام GrpcExample ایجاد می‌کند. به دایرکتوری پروژه بروید، و یک ساختار از پیش‌بینی شده با تمام فایل‌های ضروری، از جمله پوشه Protos حاوی یک فایل .proto. پیش‌فرض خواهد یافت.



این فایل Protobuf یک سرویس به نام Greeter با یک متد واحد، SayHello را مشخص می‌کند. این متد یک پیام HelloRequest را می‌بیند و یک پیام HelloReply را با پیام message برگرداند.

<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

# تعریف قرارداد سرویس با Protobuf

و مکانیسم سریالیزه کردن آن عمل می‌کند (IDL) است، که به عنوان زبان تعریف رابط gRPC، اساس کارایی و انعطاف‌پذیری Protobuf، یا سرور-فشرده تعریف کنند، که طرح کلی ارتباط کلاینت-proto به توسعه‌دهندگان اجازه می‌دهد قراردادهای سرویس و ساختارهای پیام را در یک فایل Protobuf است.

یکی از نقاط قوت کلیدی Protobuf در سریالیزه کردن بسیار کارآمد آن نهفته است aProtobuf. داده‌ها را به یک فرمت باینری فشرده کدگذاری می‌کند، برخلاف فرمتهای متنی مانند JSON یا XML، که به طور قابل توجهی اندازه پیام‌ها و سربار پردازش را کاهش می‌دهد.



<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

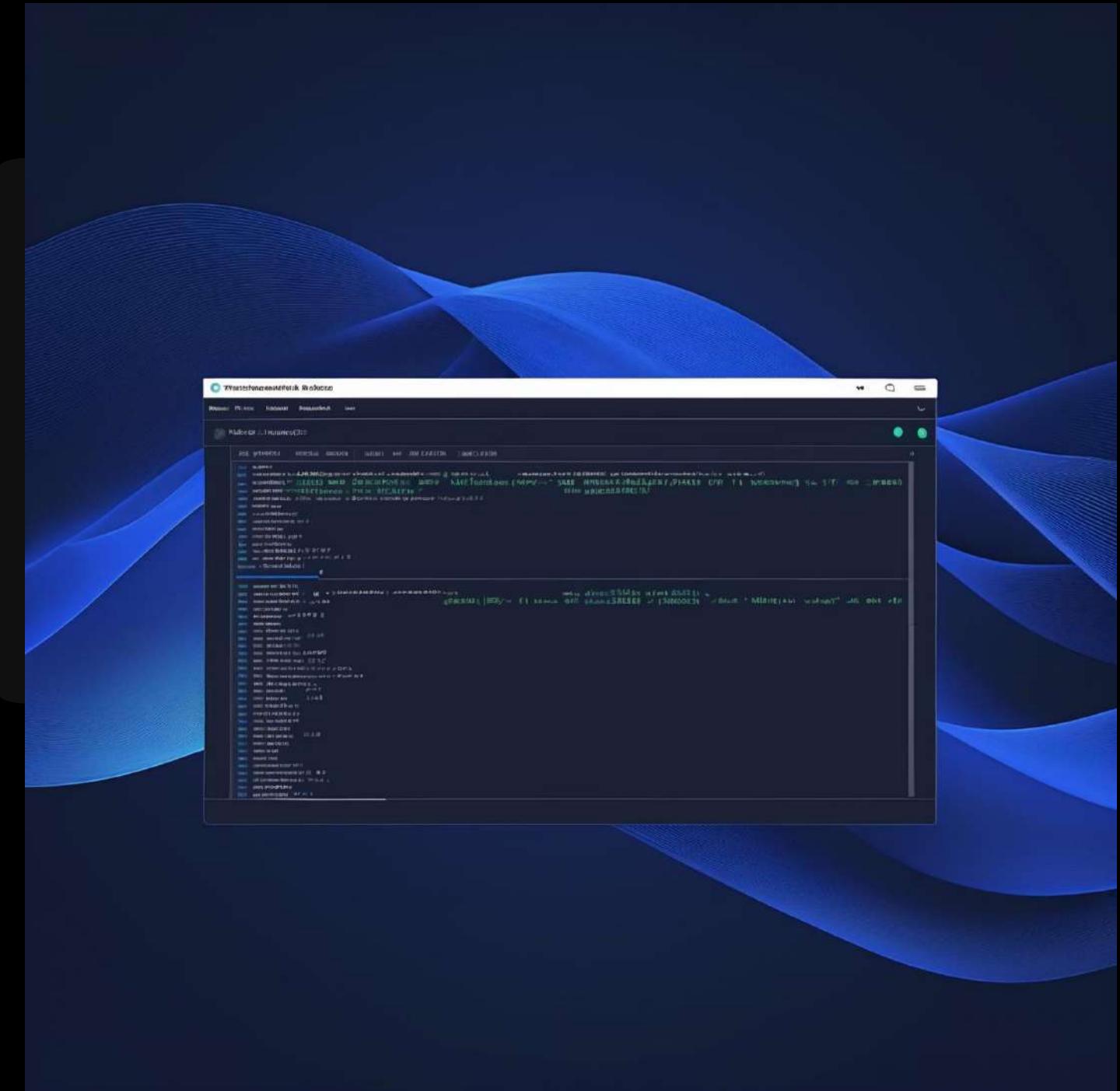
برای تعریف یک قرارداد سرویس، یک فایل .proto در دایرکتوری Protos پروره gRPC خود ایجاد کنید. به عنوان مثال، یک سرویس برای مدیریت یک لیست کارها را در نظر بگیرید، که به کاربران اجازه می‌دهد وظایف را ایجاد، بروزرسانی و حذف کنند.

# اجرای سرویس gRPC

پس از پیاده‌سازی سرویس gRPC و پیکربندی سرور، اجرای سرویس ساده است. از دستور dotnet run برای شروع برنامه استفاده کنید. اگر همه چیز به درستی تنظیم شده باشد، سرور شروع به گوش دادن به درخواست‌های ورودی gRPC و در پورت مشخص شده خواهد کرد.

به طور پیش‌فرض، ASP.NET Core یک خروجی کنسول ارائه می‌دهد که نشان می‌دهد برنامه در حال اجراست و URL که سرویس در آن قابل دسترسی است. اگر HTTPS را با HTTPS/2 پیکربندی کرده‌اید، خروجی ممکن است به این شکل باشد:

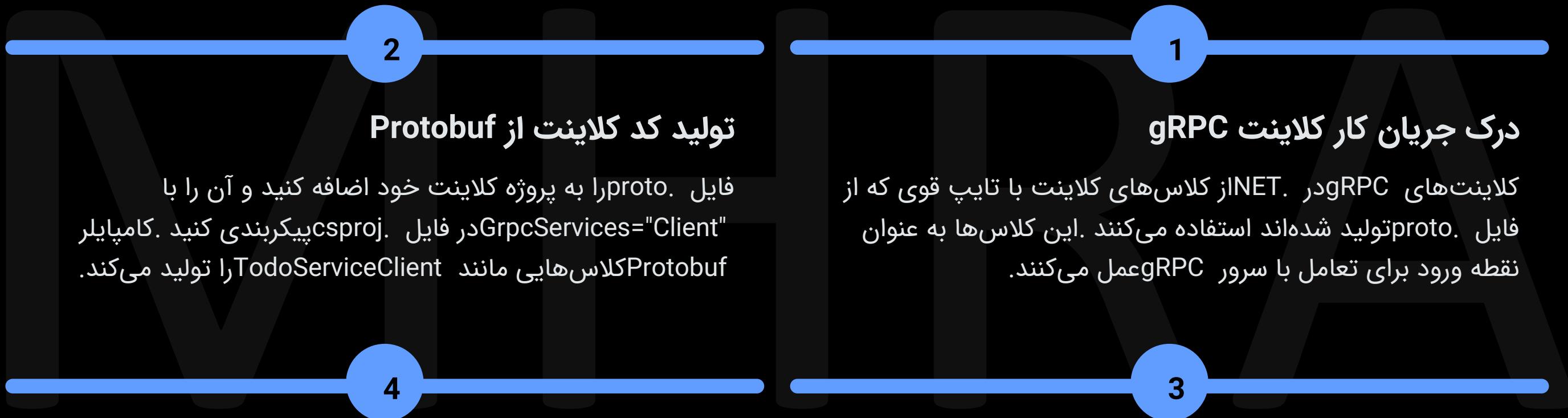
```
info: Microsoft.Hosting.Lifetime[14]      Now listening on: https://localhost:5001info:  
Microsoft.Hosting.Lifetime[14]      Now listening on: http://localhost:5000info:  
Microsoft.Hosting.Lifetime[0]       Application started. Press Ctrl+C to shut down.
```



<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

برای تأیید اینکه سرویس شما در حال اجراست، می‌توانید از ابزارهایی مانند curl یا gRPC برای ارسال درخواست به نقطه پایانی gRPC خود استفاده کنید. به عنوان مثال، اگر TodoService را پیاده‌سازی کرده‌اید، می‌توانید متodo.AddTodo را به این صورت آزمایش کنید:

# ایجاد یک کلاینت gRPC در .NET



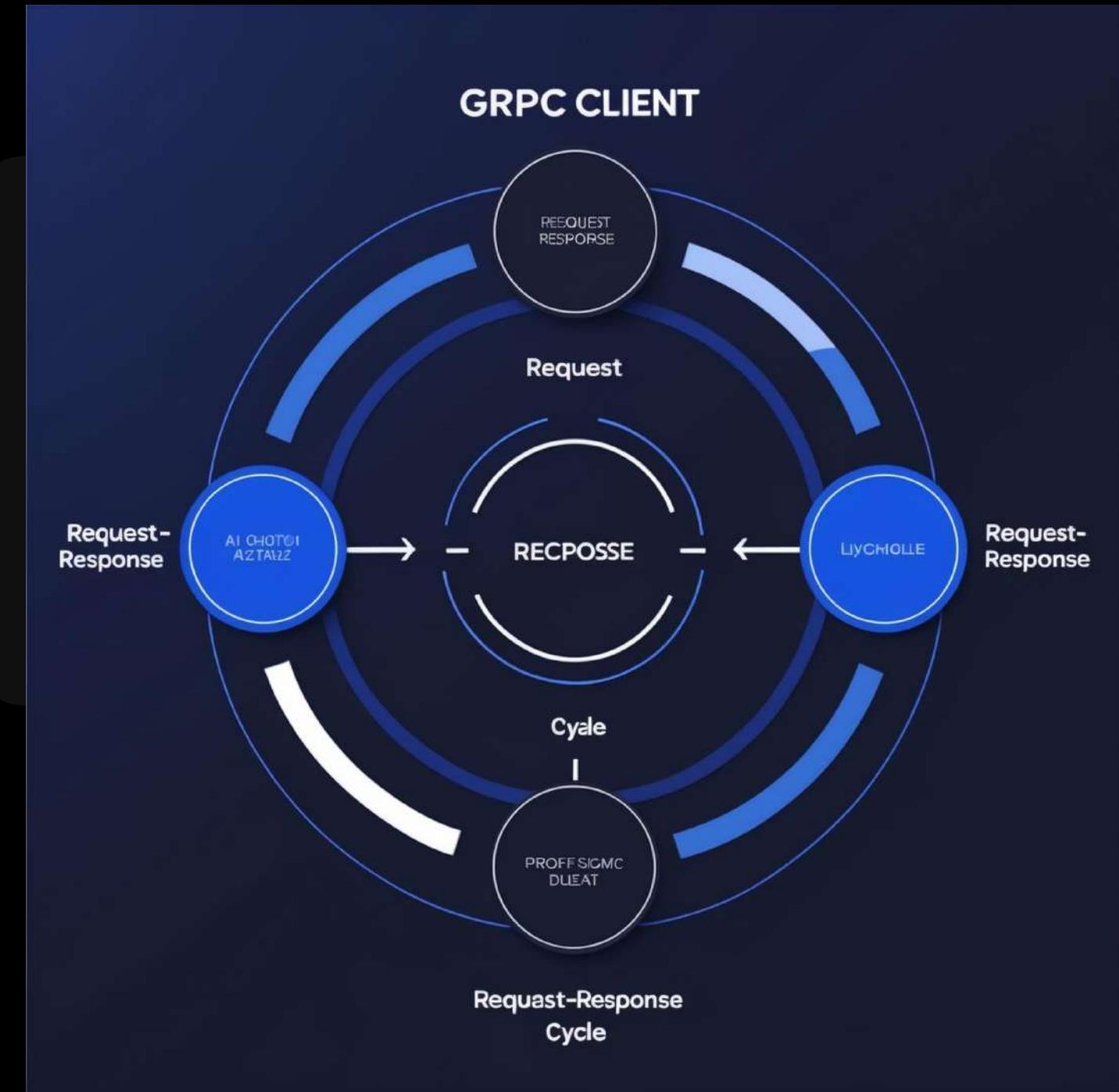
<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# درگ جریان کار کلاینت gRPC

جریان کار کلاینت gRPC در .NET می‌چرخد، با استفاده از کلاس‌های کلاینت با تایپ قوی که از فایل `.proto` تولید شده‌اند. این کلاس‌ها به عنوان نقطه ورود برای تعامل با سرور gRPC و عمل می‌کنند، منطق برای انجام فراخوانی‌های gRPC و مدیریت ارتباط شبکه را کپسوله می‌کنند.

فرآیند با کلاینت که یک کانال به سرور ایجاد می‌کند شروع می‌شود. یک کانال در gRPC و نشان‌دهنده یک اتصال به یک آدرس سرور خاص است و به عنوان پایه‌ای برای انجام فراخوانی‌های راه دور عمل می‌کند. پس از ایجاد کانال، یک شیء کلاینت با استفاده از کلاس کلاینت تولید شده نمونه‌سازی می‌شود.



<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

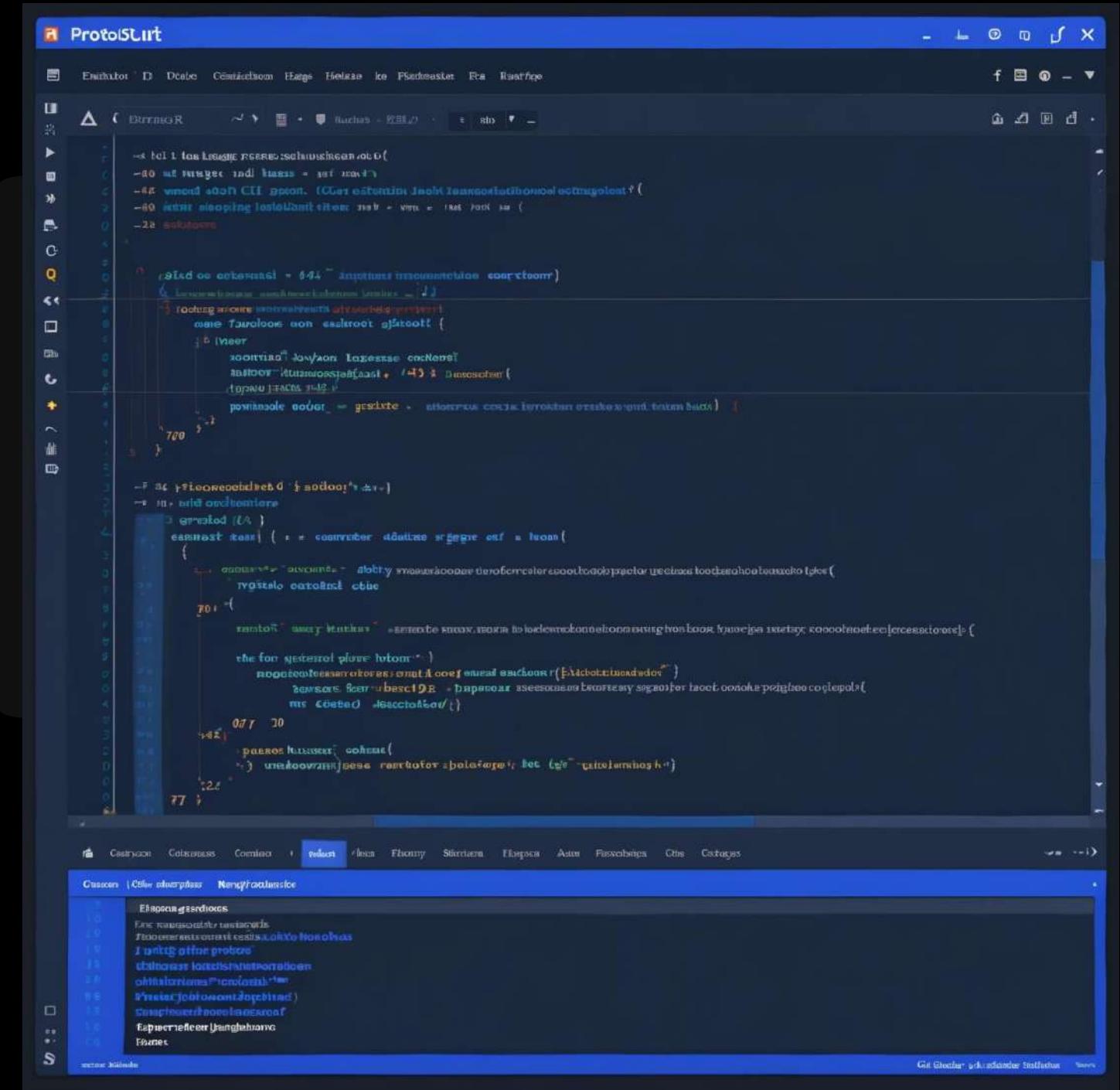
این شیء کلاینت متدهایی متاظر با gRPC‌های تعریف شده در قرارداد سرویس ارائه می‌دهد. به شما اجازه می‌دهد آنها را دقیقاً همانطور که یک متد محلی، را فراخوانی، می‌کنید، فراخوانی، کنید. به عنوان مثال، انجام یک gRPC بکتاپ، انجام یک شامل، ارسال، بیام درخواست به کلاینت با استفاده از

# تولید کد کلاینت از Protobuf

تولید کد کلاینت از Protobuf یک گام حیاتی در راه اندازی یک کلاینت RPC در .NET است. فایل .proto منبع واحد حقیقت برای تعاریف سرویس و ساختارها پیام است. ابزار dotnet-grpc کامپایل Protobuf در زمان ساخت، که کلیدی در این فرآیند هستند، به شما اجازه می‌دهند کلاس‌های #C# با تایپ قوی را تولید کنید که منطق ارتباط برای تعامل با سرور را کیپوله می‌کنند.

با افزودن فایل .proto به پروژه کلاینت خود شروع کنید. به عنوان مثال، فرض کنید فایل todo.proto از سرور، که حاوی تعریف سرویس زیر است، به پروژه کلاینت شما اضافه شده است:

```
syntax = "proto3";option csharp_namespace = "TodoApp";service TodoService { rpc AddTodo (TodoRequest) returns (TodoReply); rpc GetTodos (Empty) returns (stream TodoItem);}
```



The screenshot shows the ProtosList application window. The main area displays the generated C# code for the TodoService. The code includes imports for System, System.Collections.Generic, System.Linq, System.Threading.Tasks, and System.Text.Json. It defines a Todo class with properties for id, title, and description, and a TodoItem class with properties for id, title, and description. The TodoItem class also has a constructor and a ToString method. The TodoService interface is defined with two methods: AddTodo, which takes a TodoRequest and returns a TodoReply, and GetTodos, which returns a stream of TodoItem. The TodoRequest and TodoReply classes are also partially visible. The bottom of the window shows a navigation bar with tabs for 'Custom', 'Other categories', and 'NewProtosList', along with a search bar and other UI elements.

برای شامل کردن این فایل در پروژه کلاینت خود، آن را به دایرکتوری Protos پوشیده کنید و فایل .csproj را با موارد زیر به روزرسانی کنید:

<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

# راهاندازی کلاینت RPC.NET در

راهاندازی کلاینت برای یک سرویس RPC.NET ساده و کارآمد است. شامل ایجاد یک اتصال با سرور، ایجاد یک نمونه کلاینت و پیکربندی گزینه‌های ارتباطی است. کلاس‌های کلاینت تولید شده از تعاریف Protobuf این مراحل را ساده می‌کنند، به شما اجازه می‌دهند RPC.NET را سریعاً و با احساس بهره‌وری در برنامه خود ادغام کنید.

اولین قدم ایجاد یک GrpcChannel است، که به عنوان لینک ارتباطی بین کلاینت و سرور عمل می‌کند. کانال آدرس سرور را مشخص می‌کند و به صورت اختیاری ویژگی‌هایی مانند امنیت انتقال را پیکربندی می‌کند.

```
using Grpc.Net.Client; var channel = GrpcChannel.ForAddress("https://localhost:5001"); var client = new TodoService.TodoServiceClient(channel);
```



<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

با راهاندازی کلاینت، می‌توانید شروع به فراخوانی متدهای RPC کنید. برای فراخوانی‌های یکتایی، کلاینت متدهای ناهمگام مانند AddTodoAsync را در معرض قرار می‌دهد، که اشیاء Task را برمی‌گرداند. به عنوان مثال:

# مدیریت خطا و تلاش مجدد

## کدهای وضعیت gRPC

این کدها، مانند از کدهای وضعیت برای ارتباط خطاهای بین کلاینتها و سرورها استفاده می‌کند gRPC، به شما اجازه می‌دهند بین خطاهای قابل بازیابی و UNAVAILABLE و INVALID\_ARGUMENT غیرقابل بازیابی تمایز قائل شوید.

```
try{    var reply = await client.AddTodoAsync(new TodoRequest {        Title = "Handle Errors",        Description = "Write error handling section for gRPC chapter"    });    Console.WriteLine($"Response: {reply.Status}");}catch (RpcException ex){    Console.WriteLine($"gRPC Error: {ex.Status.StatusCode} - {ex.Message}");}
```

## مهلت‌ها و ضرب‌الاجل‌ها

می‌توانید تلاش‌های مجدد را با مهلت‌ها برای سناریوهای پیشرفته ترکیب کنید تا اطمینان حاصل کنید عملیات‌ها به طور نامحدود معلق نمی‌مانند. از شیء CallOptions برای تنظیم یک ضرب‌الاجل برای یک فراخوانی متند استفاده کنید.

## سیاست‌های تلاش مجدد

تلاش‌های مجدد برای خطاهای گذرا مانند وقفه‌های شبکه یا بار بیش از حد سرور مهم هستند gRPC. از تلاش‌های مجدد خودکار از طریق ویژگی ServiceConfig پشتیبانی می‌کند، که می‌تواند در راهاندازی کلاینت شما پیکربندی شود.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# ویژگی‌ها و الگوهای پیشرفته gRPC

توسعه دهنده‌گان را با مجموعه‌ای غنی از ویژگی‌ها و الگوهای پیشرفته توانمند gRPC می‌سازد، که به آنها امکان می‌دهد سیستم‌های بسیار کارآمد، انعطاف‌پذیر و مقیاس‌پذیر پایه فراتر می‌روند، ارتباط بلاذرنگ، gRPC این ویژگی‌ها از فراخوانی‌های .را ایجاد کنند متعادل‌سازی بار و قابلیت مشاهده را تسهیل می‌کنند.

یکی از ویژگی‌های پیشرفت‌ههای عملی و قدرتمند در gRPC استریمینگ دوطرفه است. برخلاف مدل‌های سنتی درخواست-پاسخ، استریمینگ دوطرفه به کلاینت‌ها و سرورها اجازه می‌دهد جریان‌های داده را به طور همزمان تبادل کنند. این الگو نه تنها یک مفهوم نظری، بلکه یک ابزار عملی برای برنامه‌های بلادرنگ مانند سیستم‌های چت، گزارش تله‌متري یا ابزارهای همکاري است که نیاز به ارتباط دوطرفه مداوم دارند.



# استریمینگ در gRPC: سرور، کلاینت و دوطرفه



## استریمینگ دوطرفه

کلاینت و سرور می‌توانند جریان‌ها را به طور همزمان ارسال و دریافت کنند. ارزشمند برای همکاری بلادرنگ یا سیستم‌های تعاملی مانند برنامه‌های چت.

```
public override async Task Chat(  
    IAsyncStreamReader<ChatMessage> requestStream,  
    IServerStreamWriter<ChatMessage> responseStream,  
    ServerCallContext context){ await foreach (var  
        message in requestStream.ReadAllAsync()) {  
            Console.WriteLine($"Received:  
                {message.Content}"); await  
            responseStream.WriteAsync( new ChatMessage {  
                Content = $"Echo: {message.Content}"  }); } }
```

## استریمینگ کلاینت

کلاینت یک جریان از درخواست‌ها را ارسال می‌کند، سرور یک بار پاسخ می‌دهد. ایده‌آل برای آپلودهای دسته‌ای یا وظایف تجمعی.

```
public override async Task<TodoSummary>  
    AddTodos(    IAsyncStreamReader<TodoRequest>  
    requestStream,    ServerCallContext context){  
        int count = 0;    await foreach (var request in  
            requestStream.ReadAllAsync()) {  
            _todos.Add(new TodoItem {  
                Title =  
                    request.Title, Description =  
                    request.Description, Status =  
                    "Pending"      });    count++; }  
        return new TodoSummary { TotalTodos = count }; }
```

## استریمینگ سرور

کلاینت یک درخواست واحد ارسال می‌کند، سرور یک جریان از پاسخ‌ها را برمی‌گرداند. ایده‌آل برای داشبوردهای بلادرنگ یا فیدهای دائمی.

```
public override async Task GetTodos(    Empty  
    request,    IServerStreamWriter<TodoItem>  
    responseStream,    ServerCallContext context){  
        foreach (var todo in _todos) {  
            await responseStream.WriteAsync(todo); } }
```

# اینترسپتورها برای نگرانی‌های فراگیر

اینترسپتورها در RPC یک مکانیسم قدرتمند برای مدیریت نگرانی‌های فراگیر، مانند لایکینگ، احراز هویت و متريک‌ها، بدون شلوغ کردن منطق اصلی سرویس شما ارائه می‌دهند. آنها در سطح چارچوب عمل می‌کنند، درخواست‌ها و پاسخ‌ها را هنگام عبور از خط لوله RPC ورته‌گیری می‌کنند.

در .NET، ایجاد یک اینترسپتور شامل زیرکلاس‌سازی Interceptor برای منطق همه منظوره، که می‌تواند هم به سرور و هم به کلاینت اعمال شود، یا برای منطق خاص سرور یا کلاینت است به عنوان مثال، اینجا نحوه پیاده‌سازی یک اینترسپتور سمت سرور برای ثبت تمام درخواست‌های ورودی است:

```
public class LoggingInterceptor : Interceptor{    private readonly ILogger<LoggingInterceptor> _logger;
public LoggingInterceptor(ILogger<LoggingInterceptor> logger) {    _logger = logger; }    public override async Task<TResponse> UnaryServerHandler<TRequest, TResponse>(
TRequest request,
ServerCallContext context,
UnaryServerMethod<TRequest, TResponse> continuation) {
_logger.LogInformation($"Received request: {typeof(TRequest).Name}");    var response = await
continuation(request, context);    _logger.LogInformation($"Sent response: {typeof(TResponse).Name}");
return response; }}
```



<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

برای اعمال این اینترسپتور، آن را در پیکربندی سرور Program.cs در تثبیت کنید:

# متعادل‌سازی بار و کشف سرویس

متعادل‌سازی بار و کشف سرویس برای مقیاس‌پذیری سرویس‌های gRPC و سیستم‌های توزیع‌شده حیاتی هستند. با توزیع درخواست‌های کلاینت در میان چندین نمونه سرور، متعادل‌سازی بار اطمینان می‌دهد که هیچ سرور واحدی بیش از حد بارگذاری نمی‌شود، قابلیت اطمینان و عملکرد را بهبود می‌بخشد. کشف سرویس این را تکمیل می‌کند با شناسایی پویای نمونه‌های سرور موجود، که به کلاینت‌ها اجازه می‌دهد با تغییرات در محیط، مانند استقرارهای جدید یا خرابی‌های سرور، سازگار شوند.

در gRPC، متعادل‌سازی بار می‌تواند به صورت سمت کلاینت یا سمت سرور پیکربندی شود. متعادل‌سازی بار سمت کلاینت معمولاً استفاده می‌شود زیرا وابستگی به زیرساخت خارجی را کاهش می‌دهد و از قابلیت‌های داخلی gRPC و استفاده می‌کند.



<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

یکی از این قابلیت‌ها سیاست متعادل‌سازی بار round-robin کارآمد است که از میان لیستی از نقاط پایانی سرور چرخش می‌کند، عملکرد بهینه را تضمین می‌کند.  
برای راهاندازی این آدرس‌های سرور را در یک پیکربندی StaticResolverFactory تعریف کنید:

# ایمنسازی ارتباطات gRPC

## برای ارتباطات رمزگذاری شده TLS

سنگ بنای ارتباطات رمزگذاری شده در TLS اطمینان می دهد که داده های gRPC منتقل شده بین کلاینت ها و سرورها محترمانه و به طور غیرقابل دستکاری باقی می ماند. gRPC را تشویق TLS قویاً استفاده از gRPC پیش فرض، می کند.

## محافظت در برابر تهدیدات رایج

محافظت از برنامه های gRPC در برابر تهدیدات رایج برای حفظ امنیت و قابلیت اطمینان سرویس های شما حیاتی است. سیستم های توزیع شده با بسیاری از آسیب پذیری ها، از جمله دسترسی غیرمجاز، رهگیری داده و سوء استفاده از منابع مواجه هستند.



## مکانیسم های احراز هویت

احراز هویت برای ایمنسازی سرویس های gRPC حیاتی است، اطمینان می دهد که فقط کلاینت های مجاز می توانند به داده های حساس دسترسی داشته باشند یا اقدامات خاصی را انجام دهند gRPC. از مکانیسم های مختلف احراز هویت، از جمله سیستم های مبتنی بر توکن، کلید های API و TLS پشتیبانی می کند.

## مجوز و کنترل دسترسی

مجوز و کنترل دسترسی برای اطمینان از اینکه کاربران یا کلاینت های احراز هویت شده فقط می توانند به منابع و عملیاتی که مجاز به استفاده از آنها هستند دسترسی داشته باشند، حیاتی است. در .NET، gRPC و به طور یکپارچه با ASP.NET Core چارچوب مجوز ادغام می شود.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# تست و اشکالزدایی برنامه‌های gRPC



## تست یکپارچگی جریان‌های کاری gRPC

تست یکپارچگی برای جریان‌های کاری gRPC یک گام حیاتی است که عملکرد صحیح تمام اجزای سرویس شما، از جمله سرور gRPC، کلاینت‌ها و هر وابستگی زیربنایی مانند پایگاه‌های داده یا API‌های خارجی را تضمین می‌کند. برخلاف تست‌های یکپارچگی، تست‌های واحد بر اجزای فردی با جداسازی آنها از وابستگی‌هایشان تمکز می‌کنند.



## تست واحد سرویس‌ها و کلاینت‌های gRPC

تست واحد سرویس‌ها و کلاینت‌های gRPC اطمینان از قابلیت اطمینان منطق کسب و کار برنامه شما را در حین حفظ رفتار تمیز و قابل پیش‌بینی تضمین می‌کند. برخلاف تست‌های یکپارچگی، تست‌های واحد بر اجزای فردی با جداسازی آنها از وابستگی‌هایشان تمکز می‌کنند.



## نظارت و پروفایل عملکرد

نظارت و پروفایل عملکرد برای حفظ برنامه‌های gRPC و کارآمد و قابل اعتماد ضروری است. با درک متريک‌های کلیدی عملکرد، مانند تأخیر درخواست، توان عملیاتی و استفاده از منابع، می‌توانید گلوگاه‌ها را شناسایی کنید و سرویس‌های خود را بهینه کنید.



## اشکالزدایی مشکلات رایج gRPC

اشکالزدایی برنامه‌های gRPC می‌تواند به دلیل وابستگی آنها به HTTP/2، Protobuf و سریالیزه کردن داده باینری چالش‌برانگیز باشد. شناسایی و حل مشکلات رایج نیاز به ابزارهای تشخیصی، لاگینگ مؤثر و تست ساختاریافته دارد.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# نتیجه‌گیری: آینده gRPC در .NET

تبدیل شده .NET به سرعت به یک فناوری اساسی در اکوسیستم gRPC است، با پشتیبانی قوی و ادغام یکپارچه که آن را به یک انتخاب طبیعی، می‌توانیم انتظار .NET با ادامه تکامل برای توسعه‌دهندگان می‌سازد حتی قوی‌تر شود، با ابزارها و gRPC داشته باشیم که پشتیبانی از ویژگی‌های جدیدی که توسعه را ساده‌تر و کارآمدتر می‌کنند.

روندهای آینده در gRPC احتمالاً شامل ادغام بهتر با فناوری‌های بومی ابر، پشتیبانی بهبود یافته برای محیط‌های حافظه محدود مانند WebAssembly، و ابزارهای پیشرفته‌تر برای نظارت و مدیریت سرویس‌های gRPC خواهد بود. با تکامل اکوسیستم میکروسرویس، gRPC به احتمال زیاد نقش حتی مهم‌تری به عنوان چسب ارتباطی بین سرویس‌ها ایفا خواهد کرد.

برای توسعه‌دهندگان .NET، سرمایه‌گذاری در یادگیری gRPC یک تصمیم ارزشمند است. مزایای عملکردی آن، قابلیت همکاری بین زبان‌ها و پشتیبانی قوی در چارچوب آن را به یک مهارت ضروری برای ساخت سیستم‌های توزیع شده مدرن تبدیل می‌کند. با ادامه رشد اکوسیستم gRPC، توسعه‌دهندگانی که با این فناوری آشنا هستند در موقعیت خوبی برای ساخت نسل بعدی برنامه‌های مقیاس‌پذیر، کارآمد و قابل اعتماد قرار خواهند داشت.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# کار با وب‌هوک‌ها

وب‌هوک‌ها ارتباطات برنامه‌ها را متحول کردند و جایگزینی زیبا و مبتنی بر رویداد برای مکانیسم‌های سنتی نظرسنجی ارائه می‌دهند. به جای بررسی مکرر به روزرسانی‌ها، وب‌هوک‌ها به سیستم‌ها امکان می‌دهند اعلان‌های بلادرنگ را هر زمان که رویدادهای مهمی رخ می‌دهد ارسال کنند، که به طور قابل توجهی تأخیر را کاهش داده و منابع را حفظ می‌کند.



# Webhook

## مقدمه‌ای بر وب‌هوک‌ها

### مزایای اکوسیستم داتنت

در اکوسیستم داتنت، وب‌هوک‌ها حتی جذاب‌تر می‌شوند. با ویژگی‌های پیشرفته و بهبودهای نحوی C#، پیاده‌سازی وب‌هوک‌ها اکنون در دسترس‌تر و کارآمدتر از همیشه است.

### قهرمانان ناشناخته ارتباطات مدرن

وب‌هوک‌ها در حال انقلاب آرام در نحوه تعامل برنامه‌ها در زمان واقعی هستند. آنها مکانیسمی ساده اما قدرتمند ارائه می‌دهند: به جای درخواست مکرر به روزرسانی‌ها (مانند نظرسنجی)، به سیستم‌ها اجازه می‌دهند به روزرسانی‌ها را به صورت فعال هنگام وقوع یک رویداد ارسال کنند.

### نقش حیاتی در معماری مدرن

وب‌هوک‌ها به عنوان چسبی عمل می‌کنند که اجزای مختلف را به یک معماری منسجم و مبتنی بر رویداد متصل می‌کند، خواه در حال هماهنگی میکروسرویس‌ها، مدیریت اعلان‌ها یا فعال‌سازی ادغام‌های بلادرنگ باشد.



Event-driven communication between applications

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# وبهوك چيست؟ بررسی وبهوكها

وبهوكها ممکن است مانند يك اصطلاح مُد روز به نظر برسند، اما آنها نشان دهنده تغیيری اساسی در نحوه ارتباط برنامه های مدرن هستند. در ساده ترین حالت، وبهوكها فراخوانی های HTTP هستند: مکانیسمی سبک و مبتنی بر رویداد که در آن يك برنامه داده های بلادرنگ را به برنامه دیگري از طریق يك URL خاص هر زمان که رویدادی رخ می دهد، ارسال می کند.

این مفهوم به ظاهر ساده، مشکل قابل توجهی را حل می کند - اجتناب از ناکارآمدی نظرسنجی مداوم. به جای اينکه يك برنامه مرتباً بپرسد، "آيا چيزی تغيير کرده است؟" وبهوكها به سیستم اجازه می دهند اعلام کند، "این چيزی است که اتفاق افتاده است." این ارتباط فعال به شما کنترل می دهد، تأخیر و سربار منابع را کاهش می دهد، و سیستم ها را کارآمدتر و پاسخگو تر می کند.



<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

# اکوسیستم وب‌هوك: فرستنده‌ها و گیرنده‌ها



## محموله

فرستنده یک محموله، معمولاً در قالب JSON، آماده می‌کند که جزئیات مربوط به رویداد را در بر می‌گیرد. سپس یک درخواست HTTP به یک URL از پیش‌پیکربندی شده که توسط گیرنده ارائه شده است، ارسال می‌کند.

## گیرنده

گیرنده برنامه یا سرویسی است که این اعلان‌ها را مصرف می‌کند، پردازش می‌کند و بر اساس داده‌های ورودی عمل می‌کند. یک گیرنده باید آماده باشد تا درخواست‌های ورودی را اعتبارسنجی کند، فرستنده را احراز هویت کند، و محموله را به طور کارآمد پردازش کند.

## فرستنده

فرستنده آغازگر است - برنامه‌ای که یک رویداد را تشخیص می‌دهد و مسئولیت اطلاع‌رسانی به طرف‌های علاقه‌مند را بر عهده می‌گیرد. کار فرستنده با شناسایی رویدادهای معنادار آغاز می‌شود. برای مثال، یک پلتفرم تجارت الکترونیک ممکن است یک وب‌هوك را هنگام تغییر وضعیت سفارش فعال کند.

## اتوماسیون

فرستنده‌ها و گیرنده‌ها یک خط لوله روان را تشکیل می‌دهند که ارتباط بلاذرنگ را فعال می‌کند و مداخله دستی را کاهش می‌دهد. این رابطه همزیستی سیستم‌های فردی را به یک شبکه یکپارچه از ارتباطات مبتنی بر رویداد تبدیل می‌کند.

# آغازگر گفتگو: نحوه کار وب‌هوک‌ها

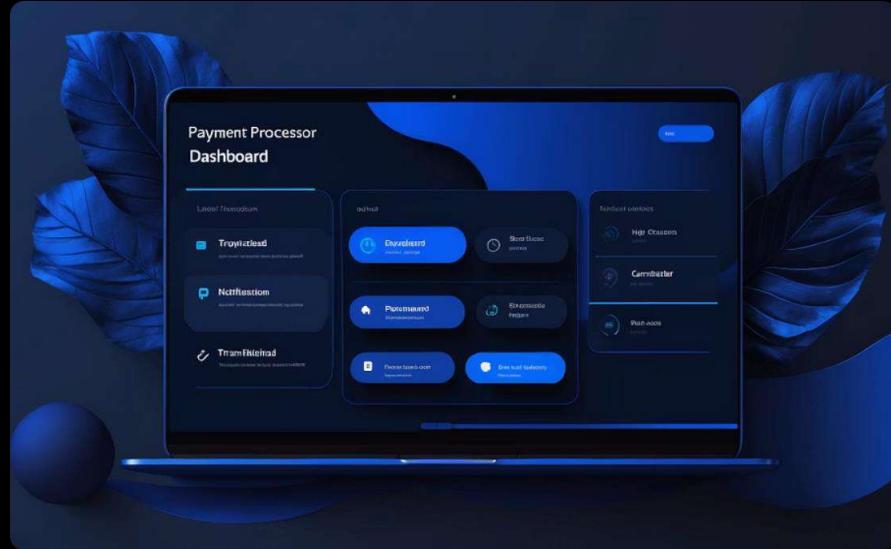
در قلب خود، یک وب‌هوک یک گفتگوی ساده اما قدرتمند بین دو سیستم است. فرستنده این گفتگو را هنگامی که یک رویداد خاص رخ می‌دهد آغاز می‌کند - فکر کنید مانند گفتن، "هی، چیزی همین الان اتفاق افتاد!" این کار با ارسال یک درخواست URL به یک HTTP POST تعیین شده که توسط گیرنده ارائه شده است، انجام می‌شود.

مهموله این درخواست حاوی تمام جزئیاتی است که گیرنده برای درک رویداد و تصمیم‌گیری در مورد اقدام بعدی نیاز دارد. این رویکرد فعال نیاز به نظرسنجی مداوم را از بین می‌برد، که وب‌هوک‌ها را به مکانیسمی بسیار کارآمد برای ارتباط بلادرنگ تبدیل می‌کند.



<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

# وبهوكها در دنياى واقعى: موارد استفاده و مثالها



## پردازش پرداخت

پردازشگرهای پرداخت مانند Stripe از وبهوكها برای هشدار به کسبوکارها در مورد تراکنشهای موفق، پرداختهای ناموفق، یا بهروزرسانیهای اشتراک استفاده میکنند. این اعلانهای بلادرنگ، که توسط وبهوكها تسهیل میشوند، نقش مهمی در همگامسازی سیستمها ایفا میکنند، و بدین ترتیب تجربه کاربر و کارایی عملیاتی را بهبود میبخشند.

## پلتفرم‌های تجارت الکترونیک

پلتفرم‌های تجارت الکترونیک مانند Shopify به شدت به وبهوكها برای اطلاع‌رسانی به فروشندهان در مورد رویدادهای مهم، مانند سفارش‌های جدید یا تغییرات موجودی، متکی هستند. هنگامی که سفارشی ثبت میشود، یک وبهوك داده‌ها را به سیستم فروشنده ارسال میکند، اطمینان می‌دهد که جریان کار پردازش سفارش آنها بدون هیچ تأخیری شروع می‌شود.

## توسعه و استقرار مداوم

تصور کنید که یک کامیت جدید را به مخزنی در GitHub فشار داده‌اید. بلافاصله، خط لوله CI/CD شما به کار می‌افتد، به لطف یک وبهوك که فرآیند ساخت و استقرار را فعال می‌کند. این اتوماسیون بدون درز، که توسط وبهوكها تغذیه می‌شود، نه تنها مداخله دستی را حذف می‌کند بلکه به شما امکان می‌دهد چرخه‌های توسعه را سریع و روان نگه دارید.

این مثالها به وضوح تنوع وبهوكها را در صنایع و برنامه‌های مختلف نشان می‌دهند. خواه در مورد ساخت یک سیستم اعلان، همگامسازی پایگاه‌های داده، یا ادغام با API‌های شخص ثالث باشد، وبهوكها کلید ایجاد برنامه‌های متصل و پاسخگو هستند.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# ایجاد یک گیرنده وب‌هوك در ASP.NET Core

3

## آماده‌سازی برای دنیای واقعی

در پایان این بخش، شما به خوبی آماده خواهید بود تا با اطمینان سناریوهای وب‌هوك دنیای واقعی را مدیریت کنید، و به طور مؤثر برنامه ASP.NET Core خود را به یک گیرنده وب‌هوك شایسته تبدیل کنید. بباید شروع کنیم و ببینیم چگونه انجام می‌شود.

2

## چالش‌های پیاده‌سازی

در حالی که مفهوم دریافت یک درخواست HTTP POST ممکن است ساده به نظر برسد، پیاده‌سازی یک گیرنده وب‌هوك قابل اعتماد کاری پیچیده است که شامل رسیدگی به ملاحظات کلیدی مانند امنیت، مقیاس‌پذیری و مدیریت خطأ است. از احراز هویت فرستنده‌ها تا تجزیه محموله‌ها و پاسخ مناسب، هر مرحله برای اطمینان از ادغام یکپارچه برنامه شما با سیستم‌های خارجی ضروری است.

1

## پایه‌ای قوی

به عنوان پایه‌ای عالی ASP.NET Core برای تبدیل تئوری به عملکرد عمل می‌کند، این به ویژه در ایجاد یک گیرنده وب‌هوك گیرنده، اساساً یک نقطه پایانی برای اعلان‌های رویداد از سیستم‌های خارجی، یک نگهبان است که بیش از یک در است درخواست‌های ورودی را اعتبارسنجی، پردازش و به آنها پاسخ می‌دهد.

# گوش دادن: راهاندازی گیرنده وب‌هوك شما

هنگامی که صحبت از کاربردهای دنیای واقعی وب‌هوك‌ها می‌شود، فایده آنها در سناریوهایی که نیاز به بهروزرسانی‌های بلاذرگ و ادغام‌های یکپارچه دارند، بیشتر مشهود است. بیایید یک مثال عملی را در نظر بگیریم: راهاندازی یک گیرنده وب‌هوك برای یک درگاه پرداخت مانند Stripe.

تصور کنید برنامه شما نیاز به مدیریت اعلان‌ها برای رویدادهایی مانند پرداخت‌های موفق یا بهروزرسانی‌های اشتراک دارد. با ASP.NET Core، فرآیند راهاندازی گیرنده مستقیم و کارآمد است، اطمینان می‌دهد که شما متصل و درگیر با بهروزرسانی‌های بلاذرگ برنامه خود باقی می‌مانید.

```
[ApiController][Route("api/[controller]")]public class WebHookController : ControllerBase{    [HttpPost]  
    public IActionResult HandleWebHook([FromBody] WebHookPayload payload) {        if (payload == null) {            return BadRequest("Invalid payload");        }        // Process the payload based on the event type  
        switch (payload.EventType) {            case "payment_success":  
                ProcessPaymentSuccess(payload);                break;            case "subscription_updated":  
                ProcessSubscriptionUpdate(payload);                break;            default:                return BadRequest("Unknown event type");        }        return Ok();    }    private void ProcessPaymentSuccess(WebHookPayload payload) {        // Custom logic for handling payment success  
        Console.WriteLine($"Payment received: {payload.Data}");    }    private void ProcessSubscriptionUpdate(WebHookPayload payload) {        // Custom logic for handling subscription updates  
        Console.WriteLine($"Subscription updated: {payload.Data}");    }}}
```



<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

# نقشه‌برداری سیگنال‌ها : پیکربندی مسیرها و نقاط پایانی

</>

۹۵

## تعریف مسیرهای اختصاصی

در یک کنترل‌کننده ASP.NET Core، می‌توانید از ویژگی‌های مسیر برای تعریف یک نقطه پایانی واضح و قابل دسترسی استفاده کنید. این یک نقطه پایانی در `https://yourdomain/api/webhooks/paymentwebhook` است. این فرآیند نه تنها انعطاف‌پذیر، بلکه به لطف قابلیت‌های قوی سیستم‌های خارجی آسان می‌کند تا اعلان‌های رویداد را تحويل دهند.

مسیریابی و پیکربندی نقطه پایانی اساس هر گیرنده وب‌هوک است، اطمینان می‌دهد که درخواست‌های ورودی به کنترل‌کننده‌های مناسب هدایت می‌شوند. در ASP.NET Core، این فرآیند نه تنها انعطاف‌پذیر، بلکه به لطف قابلیت‌های قوی مسیریابی آن، شهودی است.



## ایمن‌سازی نقاط پایانی

ایمن‌سازی نقاط پایانی خود با اجبار HTTPS، اعتبارسنجی اصالت فرستنده، و فیلتر کردن ترافیک از طریق میان‌افزار یا ویژگی‌ها بسیار مهم است. این رویکرد مسئولانه به طراحی مسیرها و نقاط پایانی شما، یک چارچوب مقیاس‌پذیر آماده برای ادغام با دنیای متنوع و پویای وب‌هوک‌ها ایجاد می‌کند.

## تفکیک انواع وب‌هوک

در سناریوهایی با چندین نوع یا ارائه‌دهنده وب‌هوک، باید بین آنها تمایز قائل شوید ASP.NET Core. به شما امکان می‌دهد مسیرهای متمازیز را به کنترل‌کننده‌ها یا اقدامات جداگانه نگاشت کنید. هر کنترل‌کننده یک نوع وب‌هوک خاص را مدیریت می‌کند، به وضوح نگرانی‌ها را جدا می‌کند.

```
[ApiController][Route("api/webhooks/{provider}")]public class DynamicWebHookController : ControllerBase{ [HttpPost] public IActionResult Receive(string provider, [FromBody] WebHookPayload payload) { Console.WriteLine($"Provider: {provider}, Event: {payload.EventType}"); return Ok(); }}
```

htt

<https://github.com/MohamadHoseinRoohiAmini>

# صحبت کردن : مدیریت و ایمنسازی درخواست‌های وب‌هوک ورودی

3

## مدیریت خطاهای و ثبت وقایع

در نهایت، مدیریت خطای و ثبت وقایع را برای ثبت مشکلاتی مانند محموله‌های بدشکل یا خطاهای پردازش پیاده‌سازی کنید. برای مثال:

```
try{    await ProcessPayloadAsync(payload);  
    _logger.LogInformation("Payload processed  
    successfully.");    return Ok();}catch  
(Exception ex){    _logger.LogError(ex,  
    "Failed to process payload.");    return  
StatusCode(500, "Internal Server Error");}
```

2

## افزایش امنیت

ایمنسازی نقطه پایانی وب‌هوک شما با اجبار HTTPS برای رمزگذاری ارتباطات و جلوگیری از دستکاری شروع می‌شود. پیکربندی خود را به روز کنید تا اطمینان حاصل شود که HTTPS مورد نیاز است:

```
builderWebHost.UseUrls("https://*:5001");
```

برای جلوگیری از حملات تکرار، مهر زمانی درخواست‌های ورودی را اعتبارسنجی کنید. برای مثال، بررسی کنید که هدر مهر زمانی در یک محدوده قابل قبول، مانند پنج دقیقه گذشته، باشد.

1

## جزیه و اعتبارسنجی درخواست‌ها

با تعریف متدهای اقدام در کنترل‌کننده‌های ASP.NET Core خود برای پردازش درخواست‌های ورودی شروع کنید. برای مثال، یک گیرنده GitHub ممکن است به این شکل باشد:

```
[HttpPost]public IActionResult  
HandleGitHubEvent([FromBody] GitHubPayload  
payload){    if (payload == null ||  
string.IsNullOrEmpty(payload.Action))        {  
return BadRequest("Invalid payload");    }  
Console.WriteLine($"GitHub Event:  
{payload.Action}");    return Ok("Event  
received successfully");}
```

با ترکیب اجبار HTTPS، اعتبارسنجی امضا، بررسی‌های مهر زمانی، فیلتر کردن IP، و ثبت وقایع دقیق، گیرنده وب‌هوک شما نه تنها کاربردی بلکه بسیار قابل اعتماد و امن می‌شود. این اقدامات اطمینان می‌دهند که فقط درخواست‌های معتبر،

به موقع و قابل اعتماد پردازش می‌شوند.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# از گزارش‌ها تا اقدامات: آزمایش و اشکال‌زدایی گیرنده شما

آزمایش و اشکال‌زدایی یک گیرنده وب‌هوک برای اطمینان از رفتار مورد انتظار آن در شرایط مختلف بسیار مهم است. هنگام ساخت در دات‌نوت، ترکیب ابزارهای ثبت وقایع قوی و قابلیت‌های اشکال‌زدایی قادر به تولید فرازیند را ساده می‌کند. به شما امکان می‌دهد مشکلات را به سرعت شناسایی و حل کنید.

با فعال کردن ثبت وقایع دقیق در برنامه ASP.NET Core خود شروع کنید. از چارچوب ثبت وقایع داخلی برای ثبت تمام درخواست‌های وب‌هوک ورودی، هدرهای آنها و محموله‌ها استفاده کنید. این به تشخیص مشکلاتی مانند محموله‌های بدشکل یا هدرهای غیرمنتظره کمک می‌کند.

```
[HttpPost]public IActionResult HandleWebHook([FromBody] WebHookPayload payload){    _logger.LogInformation("Received WebHook request.  
Headers: {Headers}, Payload: {Payload}", Request.Headers, payload);    if (payload == null)    {        _logger.LogError("Payload is null");        return BadRequest("Invalid payload");    }    _logger.LogInformation("Processing event:  
{EventType}", payload.EventType);    return Ok();}
```

ثبت نقاط داده بحرانی یک عمل حیاتی در خط لوله پردازش وب‌هوک است. این قابلیت ردیابی را تضمین می‌کند و در عیب‌یابی کمک می‌کند. با این حال، برای حفظ انطباق امنیتی، از ثبت اطلاعات حساس مانند توکن‌ها یا امضاهای خودداری کنید.

آزمایش گیرنده‌های وب‌هوک اغلب شامل شبیه‌سازی سناریوهای دنیای واقعی است. Postman، یک ابزار قدرتمند، به شما امکان می‌دهد درخواست‌های HTTP POST را با محموله‌ها و هدرهای سفارشی ایجاد کنید، که روابيدادهای واقعی وب‌هوک را تقلید می‌کند. برای آزمایش پیشرفته‌تر، Ngrok، یک ابزار سرور محلی شما را به ارائه‌دهندگان وب‌هوک خارجی با تولید یک URL عمومی موقت که می‌توانید در تنظیمات وب‌هوک یک ارائه‌دهنده پیکربندی کنید، در معرض قرار می‌دهد.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# پیاده‌سازی یک فرستنده وب‌هوک

## استفاده از ابزارهای داتنت

در این بخش، ما بررسی خواهیم کرد که چگونه یک فرستنده وب‌هوک قوی را با استفاده از API‌های شبکه پیشرفته داتنت و ویژگی‌های بیانی C# پیاده‌سازی کنیم. از تشخصیص رویدادها در برنامه شما تا تحويل ایمن محموله‌ها از طریق HTTP، شما یاد خواهید گرفت که چگونه یک فرستنده قابل اعتماد، مقیاس‌پذیر و امن بسازید.

## آغاز زنجیره همکاری

این رویکرد فعال است که وب‌هوک‌ها را به ابزاری قدرتمند برای ادغام سیستم‌های توزیع شده تبدیل می‌کند. خواه در حال اطلاع‌رسانی به یک درگاه پرداخت در مورد تغییر وضعیت یا راهاندازی گردش کارها در برنامه‌های متصل باشید، فرستنده زنجیره همکاری را آغاز می‌کند.

## تبدیل به یک مشارکت‌کننده فعال

ساخت یک فرستنده وب‌هوک برنامه شما را از یک ناظر منفعل به یک مشارکت‌کننده فعال بلادرنگ تبدیل می‌کند. به عنوان یک فرستنده، شما مسئول تشخیص رویدادها، بسته‌بندی داده‌های مربوطه، و تحويل آن به گیرنده‌های ثبت شده با دقت و قابلیت اطمینان هستید.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# تنظیم صحنه: درگ نقش فرستنده

نقش فرستنده در یک سیستم وب‌هوك محوری است -آغازگر، منبع اطلاعاتی که فرآیندهای پایین‌دستی را هدایت می‌کند. یک فرستنده وب‌هوك مسئول تشخیص رویدادهای مهم در برنامه، مانند ثبت‌نام یک کاربر جدید، خرید یک محصول، یا خطای سیستم، سریالی کردن داده‌های مربوطه به یک محموله ساختاریافته، و تحویل آن به یک گیرنده ثبت شده با استفاده از یک درخواست HTTP است.

در دات‌نت، تشخیص رویدادها می‌تواند به طور یکپارچه در برنامه شما با استفاده از الگوهای مبتنی بر رویداد ادغام شود. این الگوها، مانند الگوی Observer، به شما امکان می‌دهند یک وابستگی یک به چند بین اشیاء تعریف کنید به طوری که وقتی یک شیء تغییر وضعیت می‌دهد، تمام وابستگان آن به طور خودکار مطلع و به روز می‌شوند.

```
public class OrderService{    public event EventHandler OrderPlaced;    public void PlaceOrder(Order order)    {        // Business logic for placing an order        OrderPlaced?.Invoke(this, new OrderEventArgs(order));    }}public class OrderEventArgs : EventArgs{    public Order Order { get; }    public OrderEventArgs(Order order)    {        Order = order;    }}
```

پس از تشخیص رویداد، گام بعدی ساخت محموله است. محموله باید واضح، مختصر و سازگار باشد، معمولاً به JSON سریالی می‌شود. از System.Text.Json برای عملکرد و پشتیبانی داخلی در دات‌نت استفاده کنید.

مسئولیت اصلی فرستنده تحویل قابل اعتماد محموله است. با HttpClient به روز شده دات‌نت، این کار ساده می‌شود. می‌توانید از تزریق وابستگی برای پیکربندی HttpClient و اطمینان از استفاده مجدد کارآمد استفاده کنید.  
<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

# خوشحال از ماشه: تشخیص و ایجاد رویدادها

تشخیص رویداد، عملکرد اصلی یک فرستنده وب‌هوك، بسیار مهم است. همه چیز با یک رویداد شروع می‌شود، یک اتفاق مهم در برنامه شما که چیزی مهم برای اطلاع‌رسانی به سیستم‌های خارجی را نشان می‌دهد. ادغام و شناسابی این رویدادها در سیستم وب‌هوك شما فرآیندی است که نیاز به برنامه‌ریزی دقیق و ادغام یکپارچه در منطق کسب و کار برنامه شما دارد.

در داتنت، قدرت مدیریت تشخیص رویداد در دستان شماست، زیرا شما از رویدادها و نمایندگان استفاده می‌کنید. تصور کنید در حال ساخت برنامه‌ای هستید که ثبت‌نام کاربران را پیگیری می‌کند. شما توانایی تعریف یک رویداد سفارشی و فعال کردن آن هر زمان که یک کاربر جدید ثبت‌نام می‌کند را دارید، که شما را در مرکز فرآیند قرار می‌دهد.

```
public class UserService{    public event EventHandler UserRegistered;  
    public void RegisterUser(User user)    {        // Business logic for  
        registering a user        Console.WriteLine($"User {user.Name} registered.");  
        // Raise the event        UserRegistered?.Invoke(this, new  
        UserEventArgs(user));    } } public class UserEventArgs : EventArgs{    public User User { get; }    public UserEventArgs(User user)    {        User = user;    } }
```

کلاس `UserService` منطق ثبت‌نام را در بر می‌گیرد، در حالی که رویداد `UserRegistered` اقدامات پایین‌دستی را فعال می‌کند. این جداسازی نگرانی‌ها اطمینان می‌دهد که سیستم وب‌هوك و کار اصلی جدا باقی می‌ماند. پس از تشخیص رویداد، آن را به مکانیسم ارسال وب‌هوك خود متصل کنید. این کار با اشتراک در رویداد و فراخوانی فرستنده با محموله مربوطه انجام می‌شود.  
<https://github.com/MohamadHoseinRoohiAmini>

# ساخت پیام: ساختاردهی و سفارشی‌سازی محموله‌های وب‌هوك

## فیلتر کردن برای نیازهای گیرنده

گیرنده‌ها ممکن است به تمام داده‌هایی که سیستم شما می‌تواند ارسال کند نیاز نداشته باشند. یک مکانیسم فیلتر کردن را پیاده‌سازی کنید تا به گیرنده‌ها اجازه دهد در انواع رویداد خاص مشترک شوند یا معیارهایی برای داده‌هایی که دریافت می‌کنند تعیین کنند.

```
public class WebHookSubscription{  
    public int Id { get; set; }  
    public string ReceiverUrl { get; set; }  
    public string EventType { get; set; }  
    public string FilterCriteria { get; set; } // Optional, e.g., "orderTotal >  
                                              100"}  
                                          
```

## سربالی‌سازی کارآمد

از کتابخانه‌های سربالی‌سازی مانند System.Text.Json در دات‌نوت برای سربالی‌سازی کارآمد JSON استفاده کنید. برای مثال، یک متاد ساده برای ایجاد یک محموله ممکن است به این شکل باشد:

```
public string CreatePayload(string  
eventType, object data){  
    var payload  
        = new WebHookPayload {  
            EventType = eventType,  
            Data = data  
        };  
    return  
        JsonSerializer.Serialize(payload);  
}
```

## طراحی محموله‌های خوب

هنگام کار با وب‌هوك‌ها، محموله به عنوان پیام‌رسان عمل می‌کند، حامل جزئیات ضروری رویداد از فرستنده به گیرنده است. ساخت محموله‌های با ساختار خوب و فعال‌سازی سفارشی‌سازی برای موارد استفاده خاص، تبادل داده کارآمد را تضمین می‌کند و عملکرد را با تحويل فقط آنچه گیرنده نیاز دارد، بهبود می‌بخشد.

```
public class WebHookPayload{  
    public  
        string EventType { get; set; }  
    public  
        object Data { get; set; }  
}
```

ساختاردهی و سفارشی‌سازی محموله‌ها اطمینان می‌دهد که وب‌هوك‌ها کارآمد و متناسب با نیازهای گیرنده‌ها هستند. این سربار پردازش را کاهش می‌دهد و تجربه کلی ادغام را بهبود می‌بخشد. با پیاده‌سازی

این شیوه‌ها، سیستم وب‌هوك شما به یک ابزار ارتباطی انعطاف‌پذیر و قدرتمند تبدیل می‌شود که قادر به سازگاری با نیازهای متنوع برنامه است.  
<https://www.linkedin.com/in/mhrfahimi/>  
<https://github.com/MohamadHoseinRoohiAmini>

# تحویل کالاها: ارسال درخواست‌های وب‌هوک

پس از ساخت محموله، گام بعدی تحویل آن به گیرنده با دقت و قابلیت اطمینان است. ارسال یک درخواست وب‌هوک شامل انجام یک تماس HTTP POST به یک URL از پیش‌پیکربندی شده، شامل محموله در بدن درخواست است. با HttpClient به روز شده داتنت و API‌های شبکه بهبود یافته، این فرآیند کارآمد و دوستانه برای توسعه‌دهنده است.

با پیکربندی HttpClient با استفاده از تزریق وابستگی برای عملکرد بهینه شروع کنید. این اطمینان می‌دهد که برنامه شما از یک نمونه HttpClient استفاده می‌کند، کاوش سرور اتصال:

```
builder.Services.AddHttpClient();
```

در کلاس WebHookSender خود، یک متاد برای ارسال درخواست وب‌هوک ایجاد کنید. این متاد باید شامل محموله باشد و کدهای وضعیت پاسخ را مدیریت کند تا اطمینان حاصل شود که درخواست موفق است:

```
public class WebHookSender{    private readonly HttpClient _httpClient;    public WebHookSender(HttpClient httpClient)    {        _httpClient = httpClient;    }    public async Task SendAsync(string url, string payload)    {        var content = new StringContent(payload, Encoding.UTF8, "application/json");        var response = await _httpClient.PostAsync(url, content);        if (response.IsSuccessStatusCode)        {            Console.WriteLine("WebHook delivered successfully.");        }        else        {            Console.WriteLine($"Failed to deliver WebHook: {response.StatusCode}");        }    }}
```

# ایجاد انعطاف‌پذیری: مدیریت شکست‌ها و تلاش‌های مجدد

1

## تشخیص خطاهای گذرا

شکست‌ها در هر سیستم توزیع شده اجتناب‌ناپذیر هستند. شبکه‌ها تأخیر را تجربه می‌کنند، سرورها با زمان خاموشی مواجه می‌شوند، و مشکلات گذرا اتصال را مختل می‌کنند. توانایی یک فرستنده وب‌هوك قوی برای مدیریت این شکست‌ها به طور مناسب بسیار مهم است، اطمینان می‌دهد که رویدادها در نهایت تحويل داده می‌شوند بدون اینکه سیستم یا گیرنده را تحت فشار قرار دهد.

2

## پیاده‌سازی تلاش مجدد با Polly

کتابخانه Polly یک راه حل مؤثر ارائه می‌دهد، به شما امکان می‌دهد سیاست‌های تلاش مجدد مناسب با این سناریوها را پیاده‌سازی کنید:

```
builder.Services.AddHttpClient() .AddTransientHttpErrorPolicy(policy => policy.WaitAndRetryAsync(3, retryAttempt => TimeSpan.FromSeconds(Math.Pow(2, retryAttempt))));
```

3

## ذخیره درخواست‌های ناموفق

برای سناریوهای پیشرفته‌تر، باید درخواست‌های ناموفق را ذخیره کنید و بعداً دوباره تلاش کنید. یک رویکرد ساده شامل قرار دادن درخواست‌های ناموفق در یک فروشگاه در حافظه یا پایگاه داده است. این رویکرد نه تنها اطمینان می‌دهد که هیچ درخواستی از دست نمی‌رود بلکه امکان مدیریت بهتر تلاش‌های مجدد را نیز فراهم می‌کند.

```
public class RetryQueue{    private readonly Queue _queue = new();    public void Enqueue(WebHookRequest request)    {        _queue.Enqueue(request);    }    public WebHookRequest? Dequeue()    {        return _queue.Count > 0 ? _queue.Dequeue() : null;    }}
```

# ایمنسازی وب‌هوک‌ها



<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# امضا شده، مهر و موم شده، تحویل داده شده: تأیید محموله‌ها

اطمینان از یکپارچگی و اصالت یک محموله وب‌هوک برای جلوگیری از درخواست‌های مخرب که سیستم شما را به خطر می‌اندازند، بسیار مهم است. یکی از تکنیک‌های رایج و مؤثر استفاده از امضاهای رمزنگاری است. این امضاهای رمزنگاری از اثراً انگشت دیجیتال عمل می‌کنند، تأیید می‌کنند که یک منبع قابل اعتماد محموله را ارسال کرده است و در طول انتقال دستکاری نشده است.

بسیاری از ارائه‌دهندگان وب‌هوک یک هدر امضا در درخواست‌های HTTP خود قرار می‌دهند. برای مثال، GitHub-X-Hub-Signature-256 استفاده می‌کند، در حالی که Stripe استفاده می‌کند. فرستنده این امضا را با هش کردن محموله با استفاده از یک کلید مخفی مشترک بین فرستنده و گیرنده تولید می‌کند. وظیفه شما به عنوان گیرنده محاسبه امضای مورد انتظار و مقایسه آن با امضای موجود در هدر است.

```
[HttpPost]public IActionResult HandleWebHook([FromBody] string payload){    if (!Request.Headers.TryGetValue("X-Signature", out var signatureHeader))    {        return Unauthorized("Missing signature");    }    var secret = "your_secret_key"; // Shared between sender and receiver    var expectedSignature = ComputeHmacSha256(payload, secret);    if (signatureHeader != expectedSignature)    {        return Unauthorized("Invalid signature");    }    Console.WriteLine("Signature verified successfully.");    return Ok();}private string ComputeHmacSha256(string payload, string secret){    using var hmac = new HMACSHA256(Encoding.UTF8.GetBytes(secret));    var hash = hmac.ComputeHash(Encoding.UTF8.GetBytes(payload));    return Convert.ToString(hash);}
```

<https://www.linkedin.com/in/mhramini/>  
در این مثال، محموله با کلید مخفی مشترک با استفاده از HMAC-SHA256 هش می‌شود. درخواست فقط در صورتی اصیل در نظر گرفته می‌شود که هش محاسبه شده، یک اثر انگشت دیجیتال منحصر به فرد از محموله، با امضای ارائه شده در هدر مطابقت داشته باشد. به یاد داشته باشید که کلید مخفی مشترک باید به طور امن ذخیره شود. از متغیرهای محیطی یا یک سرویس مدیریت اسرار مانند Azure Key Vault برای جلوگیری از افشا استفاده کنید.  
<https://github.com/MohamadHoseinRouhaniArmini>

# اتصالات مجاز : مدیریت کنترل دسترسی

## کلیدهای API

برای وب‌هوک‌ها با چندین فرستنده، کلیدهای API را به عنوان یک مکانیسم کنترل دسترسی اضافی در نظر بگیرید. فرستنده کلید API را در یک هدر سفارشی قرار می‌دهد، و گیرنده آن را در مقابل لیستی از کلیدهای از پیش پیکربندی شده اعتبارسنجی می‌کند:

```
[HttpPost]public IActionResult HandleWebHook(){  
if (!Request.Headers.TryGetValue("X-API-Key", out  
    var apiKey) || !IsValidApiKey(apiKey))  
    {  
        return Unauthorized("Invalid or missing API  
key");  
    }  
    Console.WriteLine("API key  
validated.");  
    return Ok();}
```

## لیست سفید IP

یک تکنیک ساده اما بسیار مؤثر لیست سفید IP است. این رویکرد دسترسی را به یک لیست از پیش تعريف شده از آدرس‌های IP اقابل اعتماد محدود می‌کند. در ASP.NET Core، می‌توانید فیلتر کردن IP را در میان افزار برای مسدود کردن درخواست‌ها از منابع غیرقابل اعتماد پیاده‌سازی کنید:

```
app.Use(async (context, next) =>{  
    var  
    allowedIPs = new[] { "192.168.1.100",  
        "203.0.113.10" };  
    var remoteIp =  
    context.Connection.RemoteIpAddress?.ToString();  
    if (!allowedIPs.Contains(remoteIp))  
    {  
        context.Response.StatusCode =  
        StatusCodes.Status403Forbidden;  
        await  
        context.Response.WriteAsync("Forbidden:  
Unauthorized IP");  
        return;  
    }  
    await  
    next();});
```

## نقش شما در کنترل دسترسی

نقش شما در کنترل دسترسی به گیرنده وب‌هوک خود بسیار مهم است، اطمینان می‌دهد که فقط سیستم‌های مجاز می‌توانند درخواست ارسال کنند. در حالی که تأیید امضای محموله یک دفاع قوی در برابر داده‌های دستکاری شده است، مدیریت شما از کنترل دسترسی لایه دیگری از امنیت را با محدود کردن اینکه چه کسی می‌تواند حتی به نقطه پایانی شما برسد، اضافه می‌کند.

# قلاوهای ایمن در عمل: ساخت یک گردش کار امن

## اعتبارسنجی منبع

بعد، منبع درخواست را با استفاده از لیست سفید IP یا کلیدهای API اعتبارسنجی کنید. لایه اول در گردش کار شما اجبار HTTPS است. این اطمینان می‌دهد که تمام ارتباطات و ب هوک رمزگذاری شده است، محافظت از محموله و هدرها در برابر رهگیری در طول انتقال را همانطور که قبلًا توضیح داده شد پیاده‌سازی کنید، اطمینان حاصل کنید که راز مشترک شما می‌توانید کنترل این اقدام امنیتی را با پیکربندی برنامه ASP.NET Core خود برای نیاز به HTTPS در دست بگیرید:

```
app.UseHttpsRedirection();
```

## ثبت وقایع و ممیزی

در نهایت، هر درخواست ورودی را برای ممیزی و اشکال‌زدایی ثبت کنید. جزئیاتی مانند URL درخواست، هدرها، و محموله (به استثنای داده‌های حساس (را برای کمک به ردیابی هر فعالیت مشکوک شامل کنید.

## جلوگیری از حملات تکرار

برای جلوگیری از حملات تکرار، مهر زمانی هر درخواست را تأیید کنید. بسیاری از ارائه‌دهندگان وب هوک یک هدر مهر زمانی، مانند X-Timestamp، را شامل می‌کنند. بررسی کنید که مهر زمانی اخیر است تا اطمینان حاصل شود که درخواست مجدد استفاده نشده است.

این تکنیک‌ها را در یک خط لوله میان افزار یا منطق کننده ترکیب کنید تا یک گردش کار یکپارچه و امن ایجاد کنید. هر درخواست باید از مراحل اعتبارسنجی، احراز هویت، و پردازش عبور کند، اطمینان حاصل شود که فقط وب هوک‌های مشروع مورد عمل قرار می‌گیرند. این رویکرد لایه‌ای از برنامه شما محافظت می‌کند و اعتماد را با سیستم‌ها و سازمان‌هایی که به گیرنده وب هوک شما متکی هستند، ایجاد می‌کند.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# مقیاس‌بندی قلاب: عملکرد و انعطاف‌پذیری

## قدرت دات‌نت

با قدرت دات‌نت و تکنیک‌های مدرن بومی ابر، مقیاس‌بندی راه حل‌های وب‌هوك شما نباید ترسناک باشد - می‌تواند به یک الگو برای کارایی و قابلیت اطمینان تبدیل شود. از بهینه‌سازی عملکرد فرستنده و گیرنده تا پیاده‌سازی استراتژی‌های انعطاف‌پذیری، ما راه حل‌هایی ارائه خواهیم داد که به شما کمک می‌کند سیستم وب‌هوك خود را برای مقیاس آماده کنید.

3

## استراتژی‌های مقیاس‌بندی

در این بخش، ما استراتژی‌هایی برای افزایش عملکرد و انعطاف‌پذیری فرستنده و گیرنده وب‌هوك شما را بررسی خواهیم کرد. از متعادل‌سازی بار و پردازش ناهمگام تا پیاده‌سازی تلاش‌های مجدد و صفاتی توزیع شده، شما یاد خواهید گرفت که چگونه سیستمی طراحی کنید که تحت استفاده سنگین رونق می‌یابد و قابلیت اطمینان را حفظ می‌کند.

2

## رشد اجتناب‌ناپذیر

با رشد برنامه شما، تقاضا برای پیاده‌سازی وب‌هوك شما نیز به طور اجتناب‌ناپذیر افزایش خواهد یافت. مقیاس‌بندی یک سیستم وب‌هوك نیاز به بهینه‌سازی عملکرد برای ترافیک بالا و اطمینان از انعطاف‌پذیری در برابر شکست‌ها و قطعی‌ها دارد. یک وب‌هوك از دست رفته می‌تواند گردش کارها را مختل کند، در حالی که یک گیرنده تحت فشار ممکن است باعث تأخیر یا خرابی شود.

1

# معناد به سرعت: بهینهسازی عملکرد

بهینهسازی عملکرد سیستم وب‌هوك شما اطمینان می‌دهد که می‌تواند حجم بالایی از درخواست‌ها را بدون کند شدن یا گلوگاه مدیریت کند. کلید، ساده‌سازی فرآیندهای ارسال و دریافت، به حداقل رساندن تأخیر و مصرف منابع در حین حفظ سطح بالایی از قابلیت اطمینان است. در داتنت، ابزارها و تکنیک‌های قدرتمند می‌توانند به شما در دستیابی به این اهداف به طور کارآمد کمک کنند.

با بهینهسازی فرستنده شروع کنید. از HttpClient به طور مؤثر با پیکربندی آن برای استفاده مجدد از طریق تزریق وابستگی استفاده کنید. این از سربار ایجاد و دفع مکرر نمونه‌های HttpClient جلوگیری می‌کند:

```
builder.Services.AddHttpClient(client =>{
    client.Timeout = TimeSpan.FromSeconds(10);});
```

در سمت گیرنده، پردازش ناهمگام می‌تواند عملکرد را به طور چشمگیری بهبود بخشد. با جدا کردن دریافت یک وب‌هوك از منطق کسب و کاری که فعال می‌کند، منابع را برای مدیریت سریع‌تر درخواست‌های ورودی آزاد می‌کنید. از صفحه‌ای پیام، مانند RabbitMQ یا Azure Service Bus، برای برونو سپاری پردازش استفاده کنید:

```
[HttpPost]public async Task ReceiveWebHook([FromBody]
    WebHookPayload payload){    await
    _messageQueue.EnqueueAsync(payload);    return
    Accepted();}
```

در این مثال، پاسخ Accepted به فرستنده اطلاع می‌دهد که وب‌هوك با موفقیت دریافت شده است، حتی اگر پردازش به صورت ناهمگام در پس‌زمینه انجام شود.

راه دیگر برای بهبود عملکرد کاهش اندازه محموله است. اگر محموله‌های وب‌هوك شما شامل داده‌های تکراری یا بیش از حد جزئیات است، ساختار محموله را ساده کنید. برای مثال، فقط نوع رویداد و یک شناسه را که گیرنده می‌تواند از آن برای دریافت جزئیات اضافی در صورت نیاز استفاده کند، شامل کنید.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

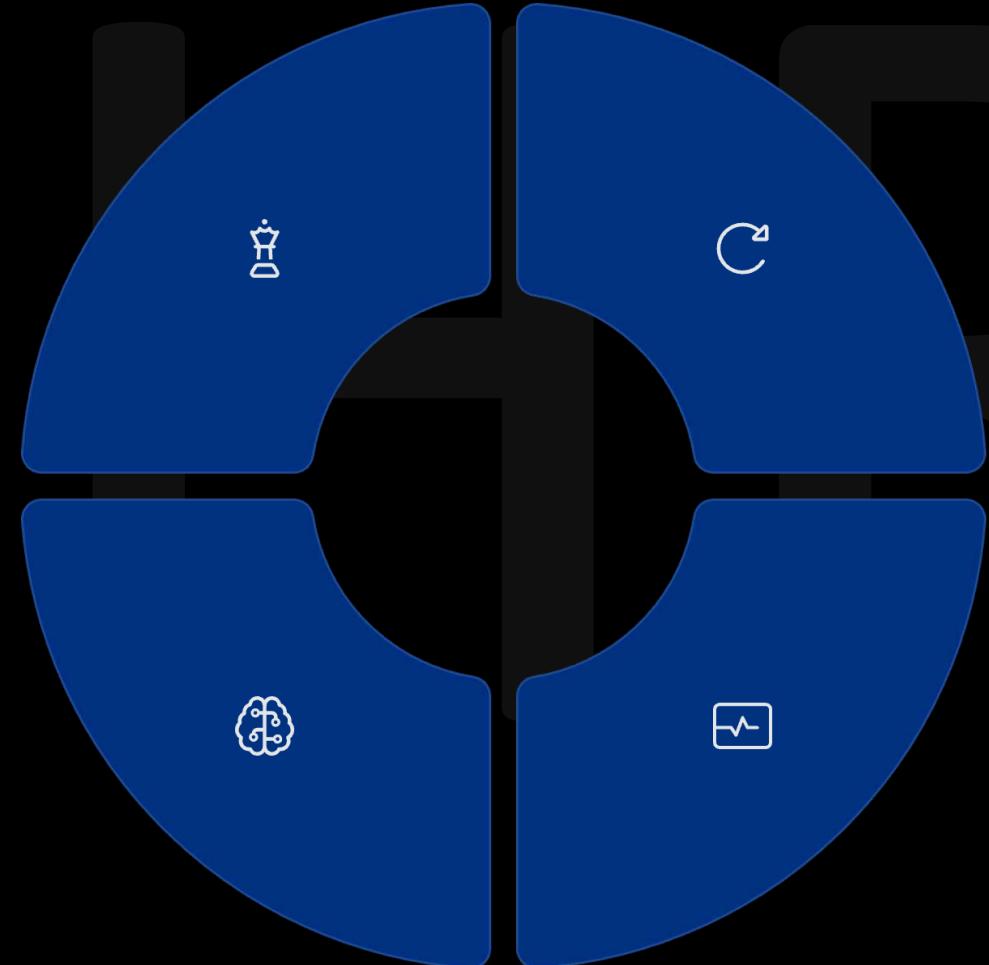
# زنده نگه داشتن قلاب: طراحی برای انعطاف‌پذیری

## پردازش ناهمگام

جنبه مهم دیگر انعطاف‌پذیری، جدا کردن دریافت وب‌هوک از پردازش است. با بروزسپاری پردازش به یک سرویس پس‌زمینه، اطمینان می‌دهید که گیرنده حتی اگر سیستم‌های پایین‌دستی کند باشند، پاسخ‌گو باقی می‌ماند. از یک صفحه پیام مانند Azure Service Bus برای ذخیره رویدادهای وب‌هوک برای پردازش ناهمگام استفاده کنید.

## مدارشکن‌ها

مدارشکن‌ها یک جزء کلیدی در محافظت از سیستم شما در برابر شکست‌های آبشاری هستند. هنگامی که یک سرویس پایین‌دستی غیرقابل دسترس می‌شود، یک مدارشکن وارد عمل می‌شود، موقتاً سیستم را از ارسال درخواست‌ها متوقف می‌کند. این مکث به سرویس زمان می‌دهد تا بهبود یابد، از آسیب بیشتر جلوگیری می‌کند.



## منطق تلاش مجدد

طراحی یک سیستم وب‌هوک انعطاف‌پذیر اطمینان می‌دهد که می‌تواند از شکست‌ها بهبود یابد و تحت شرایط نامساعد به کار خود ادامه دهد. با پیاده‌سازی منطق تلاش مجدد برای خطاهای گذرا شروع کنید. از کتابخانه Polly برای مدیریت تلاش‌های مجدد با پس‌نشینی نمایی استفاده کنید.

## ناظارت بر سلامت

سلامت سیستم وب‌هوک خود را ناظرت کنید. از ابزارهایی مانند Azure Monitor یا Prometheus برای پیگیری معیارهای کلیدی مانند نرخ موفقیت تحويل، تعداد تلاش‌های مجدد، و وضعیت مدارشکن استفاده کنید. با تنظیم هشدارها برای ناهنجاری‌ها، می‌توانید آماده باشید و قبل از اینکه مشکلات تشدید شوند، به طور فعال پاسخ دهید.

ایجاد انعطاف‌پذیری در سیستم وب‌هوک شما اطمینان می‌دهد که می‌تواند چالش‌های اجتناب‌ناپذیر محیط‌های توزیع شده را مدیریت کند. با ترکیب تلاش‌های مجدد، گردش کارهای

ناهمگام، مدارشکن‌ها، و ناظارت، راه حلی ایجاد می‌کنید که نه تنها از شکست‌ها جان سالم به در می‌برد بلکه پس از آنها روتق می‌یابد.

<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

# فراز از اصول اولیه: الگوهای پیشرفته وب‌هوک

## اعلان‌های انتخابی

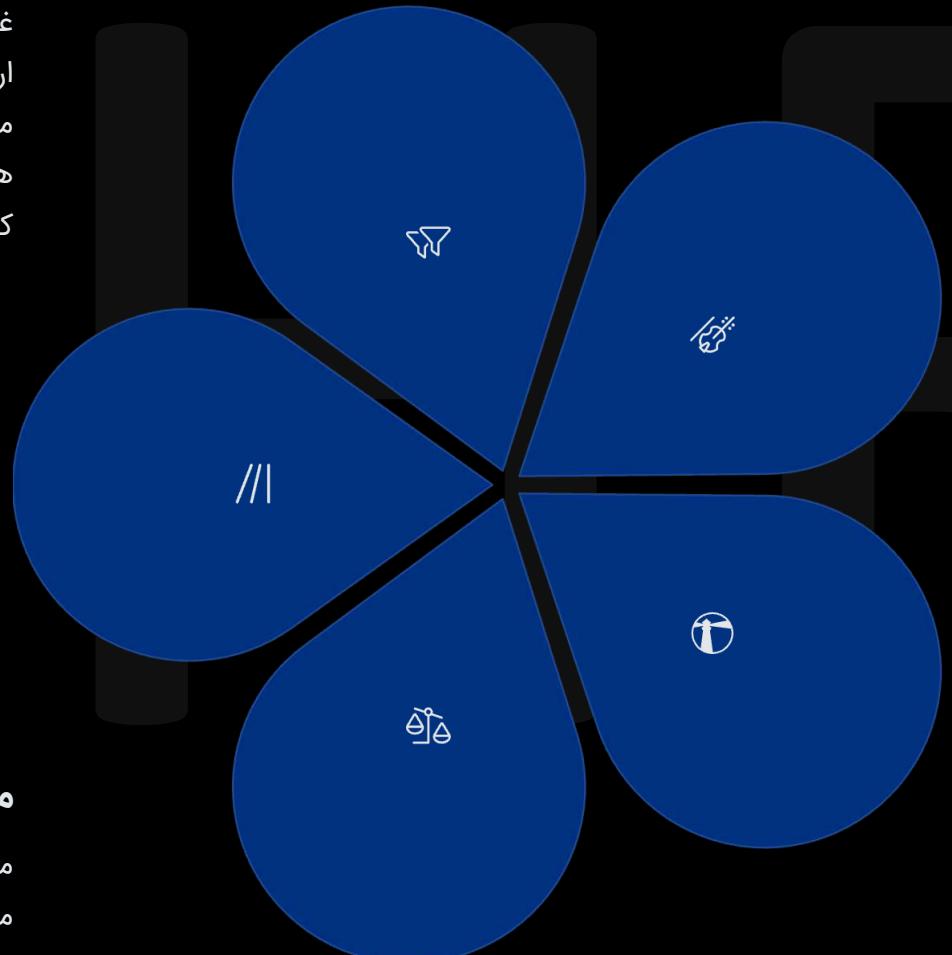
تطبیق اعلان‌های وب‌هوک با نیازهای خاص گیرنده‌ها انتقال داده‌های غیرضروری را کاهش می‌دهد و کارایی ادغام را افزایش می‌دهد. به جای ارسال همه رویدادها به همه مشترکین، فیلتر کردن پویا به گیرنده‌ها اجازه می‌دهد فقط رویدادهایی را که به آنها اهمیت می‌دهند انتخاب کنند. در همین حال، محموله‌های سفارشی اطمینان می‌دهند که آنها فقط اطلاعاتی را که نیاز دارند دریافت می‌کنند.

## قلاب‌های انعطاف‌پذیر

اطمینان از انعطاف‌پذیری با رشد پیچیدگی و تقاضای سیستم وب‌هوک شما بسیار مهم می‌شود. شکست‌ها، مانند قطعی‌های شبکه یا گیرنده‌های تحت فشار، در سیستم‌های توزیع شده اجتناب‌ناپذیر هستند. برای مدیریت مناسب این موارد، استراتژی‌هایی مانند صفحه‌ای پیام، مکانیسم‌های تلاش مجدد، و متعادل‌سازی بار را معرفی کنید. اطمینان حاصل کنید که هر وب‌هوک به طور قابل اعتماد و در مقیاس تحويل داده می‌شود.

## مقیاس‌بندی وب

مدیریت ترافیک بالا با ظرفت برای مقیاس‌بندی سیستم وب‌هوک شما بسیار مهم است. هنگامی که یک افزایش ناگهانی در رویدادها رخ می‌دهد -مانند در طول یک فروش فلش یا کمپین ویروسی- سیستم شما باید درخواست‌ها را به طور کارآمد پردازش کند بدون اینکه محموله‌ها را از دست بدهد یا منابع را تحت فشار قرار دهد.



## قلاب‌های هماهنگ شده

هماهنگی گردش کارها در چندین سرویس یک چالش رایج در سیستم‌های توزیع شده است. وب‌هوک‌ها نقش مهمی در این هماهنگی ایفا می‌کنند. در اجازه می‌دهند اقدام یک سرویس رویدادهای وابسته را در دیگران فعال کند. با زنجیره کردن وب‌هوک‌ها، می‌توانید یک خط لوله پویا و مبتنی بر رویداد ایجاد کنید که اطمینان می‌دهد هر سرویس به طور یکپارچه ارتباط برقرار می‌کند.

## ناظارت بر قلاب

قابلیت اطمینان در یک سیستم وب‌هوک به ناظارت و ثبت وقایع فعلی بستگی دارد. بدون دید به رفتار سیستم شما، مشکلاتی مانند تحويل‌های ناموفق، عملکرد کاهش یافته، یا افزایش‌های ناگهانی ترافیک غیرمنتظره می‌توانند نادیده گرفته شوند تا زمانی که به مشکلات بزرگی تبدیل شوند. با پیاده‌سازی ناظارت قوی، می‌توانید مشکلات را سریعاً تشخیص، تشخیص و حل کنید.

# نتیجه‌گیری

ما در این بحث، وب‌هوک‌ها را از مفاهیم پایه تا پیاده‌سازی‌های پیشرفته بررسی کردیم و دیدیم که چگونه این ابزارها ارتباط بی‌درنگ بین سیستم‌های توزیع شده را ممکن می‌سازند.

مباحثت کلیدی شامل ایجاد فرستنده‌ها و گیرنده‌ها در ASP.NET Core، ایمن‌سازی ارتباطات، و بهینه‌سازی برای عملکرد و انعطاف‌پذیری در برابر خطاهای بود.

با به‌کارگیری این اصول، می‌توانید راه حل‌های وب‌هوک کارآمد، ایمن، مقیاس‌پذیر و انعطاف‌پذیر بسازید که نوآوری و ارتباط یکپارچه را در توسعه مدرن ممکن می‌سازند.

# پیاده‌سازی صف پیام

صف پیام، ابزاری محوری در سیستم‌های نرم‌افزاری مدرن، ارتباط و هماهنگی قابل اعتماد، مقیاس‌پذیر و انعطاف‌پذیر بین برنامه‌ها را تسهیل می‌کند. در این ارائه، به دنیای جذاب صف پیام می‌پردازیم و بررسی می‌کنیم که چگونه .NET 8 و C# 12 به شما امکان می‌دهند از قدرت آن با طرافت و کارایی بهره ببرید.



# مقدمه‌ای بر صفحه پیام

تصور کنید شهری پر جنب و جوش با خدمات تحویل که خیابان‌ها را طی می‌کنند و هر کدام با دقت بسته‌ها را به مقصدشان هدایت می‌کنند. در سیستم‌های نرم‌افزاری، صفحه‌ای پیام نقشی مشابه دارند و به عنوان پیکهای داده عمل می‌کنند، اطمینان می‌دهند که پیام‌ها به طور قابل اعتماد و کارآمد بین اجزا تحویل داده می‌شوند.

آنها نگهبانان جریان روان ارتباطات هستند، حتی زمانی که اتفاقات غیرمنتظره مانند تأخیرها یا قطعی سیستم رخ می‌دهد. این رویکرد به سیستم‌ها امکان می‌دهد به صورت ناهمگام عمل کنند و تولیدکنندگان و مصرفکنندگان را از هم جدا کنند.



# مفاهیم اصلی صفحه پیام

در قلب صفحه پیام، مفهومی ساده اما قدرتمند نهفته است: تولیدکنندگان پیام‌ها را ایجاد می‌کنند، صفحات آنها را به طور موقت ذخیره می‌کنند، و مصرفکنندگان آنها را بازیابی و پردازش می‌کنند.

یک پیام، واحد اساسی داده‌های منتقل شده است که اغلب به صورت JSON، XML یا باینری فرمت می‌شود. می‌تواند هر چیزی از سفارش کاربر در یک سیستم تجارت الکترونیک تا اطلاعیه‌ای درباره یک وظیفه تکمیل شده را نشان دهد.

صفحات به عنوان محل نگهداری عمل می‌کنند که در آن پیام‌ها منتظر پردازش می‌مانند. در بسیاری از پیاده‌سازی‌ها، این صفحات بر اساس اصل اولین ورودی، اولین خروجی (FIFO) عمل می‌کنند، اطمینان می‌دهند که پیام‌ها به ترتیبی که ارسال شده‌اند پردازش می‌شوند.



# الگوهای ارتباطی در صفحه پیام

(۱۹)

100

## الگوی انتشار-اشتراک

به یک تولیدکننده امکان می‌دهد پیام‌ها را به چندین مشترک پخش کند، مناسب برای سناریوهایی مانند ارسال اعلان‌های بلادرنگ به کاربران. در این مدل، هر پیام می‌تواند توسط چندین مصرف‌کننده پردازش شود.

## الگوی نقطه به نقطه

یک تولیدکننده را با یک مصرف‌کننده جفت می‌کند، ایده‌آل برای وظایفی مانند پردازش کار. در این مدل، هر پیام فقط توسط یک مصرف‌کننده پردازش می‌شود.

این الگوهای اصلی به توسعه‌دهندگان امکان می‌دهند سیستم‌های ارتباطی قوی متناسب با نیازهای برنامه خود طراحی کنند. با تسلط بر این مفاهیم بنیادی، آماده خواهید بود تا موضوعات پیشرفته‌تر را مورد بررسی قرار دهید.

# نقش صفاتی پیام در برنامه‌های مدرن

## مقیاس‌پذیری

نیاز به پردازش پیام‌های بیشتر دارد؟ مصرف‌کنندگان بیشتری اضافه کنید. می‌خواهید ویژگی جدیدی را پیاده‌سازی کنید؟ بدون اختلال در کل سیستم، یک تولیدکننده یا مشترک دیگر معرفی کنید.

## انعطاف‌پذیری

با قرار دادن پیام‌ها در صفت برای پردازش بعدی، سیستم‌ها انعطاف‌پذیری به دست می‌آورند، عملکرد را حفظ می‌کنند و تقاضا را به طور مناسب مدیریت می‌کنند.

## ارتباط ناهمگام

صفاتی پیام به عنوان واسطه عمل می‌کنند و به تولیدکنندگان امکان می‌دهند بدون نگرانی در مورد زمان یا نحوه پردازش پیام‌ها توسط مصرف‌کنندگان، پیام‌ها را ارسال کنند.

این انعطاف‌پذیری، صفاتی پیام را برای میکروسرویس‌ها، معماری‌های بدون سرور و سایر سیستم‌های توزیع شده ضروری می‌سازد. صفاتی پیام به عنوان چسبی عمل می‌کنند که سیستم‌های پیچیده را به هم متصل می‌کند - اطمینان می‌دهند که آنها به طور روان عمل می‌کنند، به طور موثر مقیاس‌پذیر می‌شوند و در مواجهه با چالش‌ها انعطاف‌پذیر باقی می‌مانند.

# بررسی موارد استفاده برای صف پیام

## پلتفرم تجارت الکترونیک

در یک پلتفرم تجارت الکترونیک که هزاران سفارش را در طول یک فروش ویژه پردازش می‌کند، به جای اینکه فرآیند پرداخت متظر بررسی موجودی، پرداش پرداخت و تکمیل سفارش باشد، سیستم از صف پیام استفاده می‌کند: اطلاع‌رسانی به دنبال‌کنندگان، به‌روزرسانی فیدها و ثبت تحلیل‌ها. یک صف پیام انتشار-اشتراك این را به طور ظریف مدیریت می‌کند. هر مرحله پیامی ایجاد می‌کند که در صف برای پرداش بعدی قرار می‌گیرد، اطمینان می‌دهد که مشتریان تجربه پرداخت سریع و بدون وقفه‌ای داشته باشند.

## سیستم‌های مبتنی بر رویداد

در سیستم‌های مبتنی بر رویداد مانند سرویس‌های اعلان بلادرنگ، وقتی کاربر شبکه اجتماعی عکسی را ارسال می‌کند، برنامه چندین رویداد ایجاد می‌کند: اطلاع‌رسانی به دنبال‌کنندگان، به‌روزرسانی فیدها و ثبت تحلیل‌ها. یک صف پیام انتشار-اشتراك این را به طور ظریف مدیریت می‌کند.

## دستگاه‌های اینترنت اشیاء (IoT)

صفهای پیام در حوزه دستگاه‌های IoT ضروری هستند. دستگاه‌هایی مانند ترمومترات‌های هوشمند و سنسورها به طور مداوم داده‌هایی با نرخ‌های غیرقابل پیش‌بینی تولید می‌کنند. یک صف پیام این داده‌های ورودی را بافر می‌کند، تحويل قابل اعتماد به سرویس‌های تحلیلی را تضمین می‌کند، حتی در طول قطعی موقت شبکه.

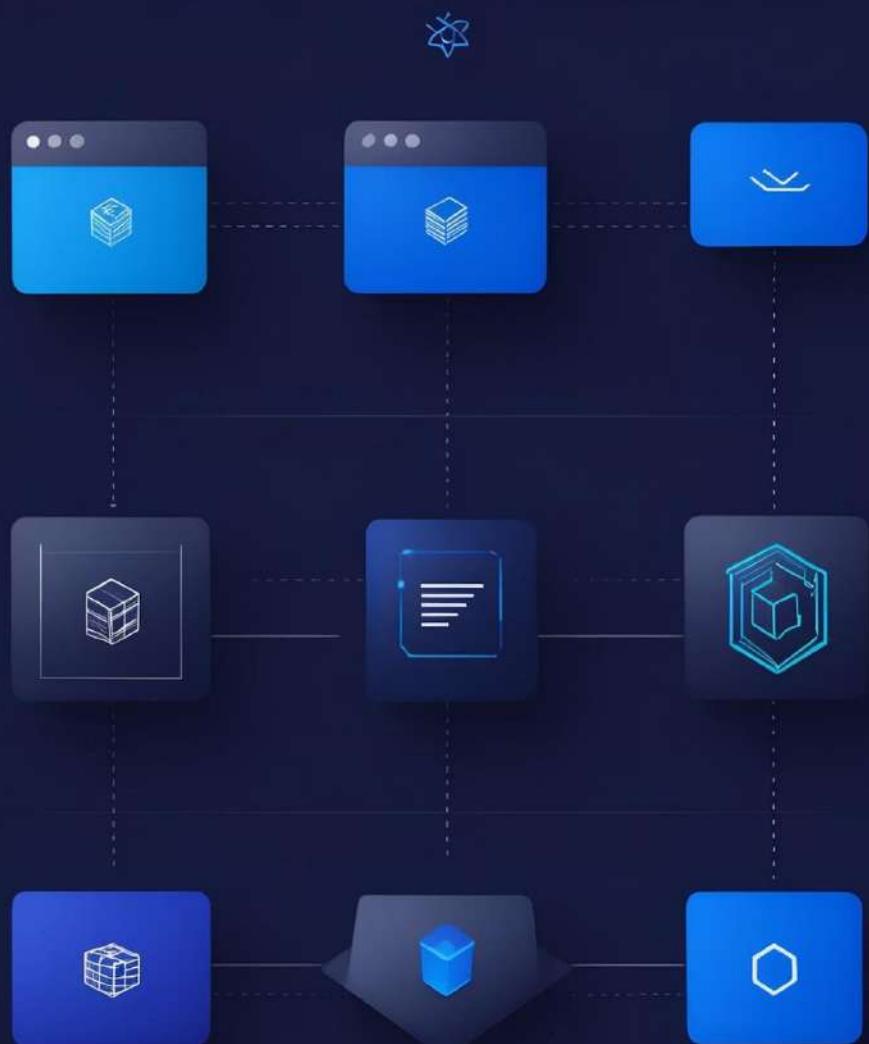
<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# Different Message Queue Queue Technologies

COMPARISON

Style Luxurious / creative professional  
Indulgæ Hglæ Coræl Feel



## بررسی فناوری‌های صفحه پیام

صفهای پیام راه حل‌های یکسان برای همه نیستند - برنامه‌های مختلف به ابزارهای متفاوتی نیاز دارند، و اکوسیستم فناوری‌های صفحه پیام طیف متنوعی از گزینه‌ها را برای برآورده کردن نیازهای متنوع ارائه می‌دهد. هر فناوری نقاط قوت منحصر به فردی را به همراه می‌آورد، از راه حل‌های سبک و متن باز گرفته تا سرویس‌های سازمانی و مبتنی بر ابر.

در این بخش، پلتفرم‌های پیشرو در صفحه پیام، از جمله Apache Kafka، RabbitMQ، Azure Service Bus و Amazon SQS را بررسی می‌کنیم. با بررسی ویژگی‌ها، نقاط قوت و مبادلات آنها، بینشی در مورد اینکه کدام ابزار برای سناریوهای خاص مناسب‌تر است، به دست می‌آورید.

# مرور فناوری‌های صفت پیام

دنیای صفحه‌های پیام به طور قابل توجهی متنوع است و طیفی از ابزارها را ارائه می‌دهد که می‌توانند برای مقیاس‌ها، معماری‌ها و نیازهای عملکردی مختلف تنظیم شوند. این انطباق‌پذیری یک ویژگی کلیدی هر فناوری صفت پیام است که ارتباط ناهمگام را در هسته خود فعال می‌کند.

2

## سرویس‌های مبتنی بر ابر

برای کسانی که سرویس‌های مدیریت شده را ترجیح می‌دهند، صفحه‌های پیام بومی ابر مانند Amazon SQS و Azure Service Bus قابلیت‌های پیامرسانی قدرتمندی را بدون سریار عملیاتی ارائه می‌دهند. هر دو سرویس با SDK‌های .NET و Java برای ارائه می‌شوند.

1

## راه حل‌های متن باز

راه حل‌های متن باز مانند Apache Kafka و RabbitMQ پیش‌نیاز استقرارهای خودمدیریتی هستند. RabbitMQ چاقوی ارتقش سوئیس صفت پیام است که انعطاف‌پذیری با ارتباط مبتنی بر AMQP، پشتیبانی از انواع مختلف تبادل و پلاگین‌هایی برای گسترش عملکرد را ارائه می‌دهد.

با درک عمیق‌تر این فناوری‌ها، خواهید دید که چگونه هر کدام می‌توانند موارد استفاده خاصی را مورد توجه قرار دهند و به طور یکپارچه در پروژه‌های برنامه‌نویسی شبکه شما جای گیرند.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# پلتفرم‌های محبوب صف پیام

## RabbitMQ: رقیب همه‌کاره

، استاندارد انعطاف‌پذیری، یکی از واسطه‌های پیام متن باز RabbitMQ به آن AMQP پایه آن بر پروتکل است که بیشترین استفاده را دارد. امکان می‌دهد از طیف متنوعی از الگوهای پیام‌رسانی پشتیبانی کند یک کتابخانه قوی و کاربرپسند برای تولید .NET ادغام بی‌درنگ آن با و مصرف پیام‌ها ارائه می‌دهد.

## Apache Kafka: قدرت جریان رویداد

اگر با حجم عظیمی از داده‌های بلاذرنگ سروکار دارید، Apache Kafka پلتفرم انتخابی است. طراحی شده برای توان عملیاتی بالا و تأخیر کم، Kafka در جریان رویداد، تجمعیع لاغ و برنامه‌های داده بزرگ می‌درخشد. کتابخانه‌های کلاینت Kafka برای .NET تولید و مصرف جریان‌ها را ساده می‌کند.

## Azure Service Bus: اسب کار سازمانی

برای توسعه‌دهندگانی که برنامه‌های سطح سازمانی می‌سازند، Azure Service Bus یک راه حل پیام‌رسانی بومی ابر با ویژگی‌های متعدد ارائه می‌دهد. از هر دو صفت و موضوع پشتیبانی می‌کند، که آن را برای الگوهای پیام‌رسانی مختلف همه‌کاره می‌سازد. با ادغام محکم در اکوسیستم Azure و یک SDK غنی برای .NET.

## Amazon SQS: سادگی در مقیاس

با رویکرد مینیمالیستی خود، یک سرویس صفاتی کاملًا Amazon SQS مدیریت شده و مقیاس‌پذیر ارائه می‌دهد که به راحتی میلیون‌ها پیام این مقیاس‌پذیری، همراه با ادغام در ثانیه را مدیریت می‌کند ، آن را به گزینه‌ای جذاب برای AWS بی‌درنگ آن با سرویس‌های سیستم‌های توزیع شده در مقیاس بزرگ تبدیل می‌کند.

# مقایسه ویژگی‌های کلیدی

## ادغام و اکوسیستم: اتصال همه چیز

ادغام یک جنبه مهم در انتخاب صفات پیام است. Apache Kafka و RabbitMQ اکوسیستم‌های متن باز قوی و کتابخانه‌های یکپارچه .NET را ارائه می‌دهند. Azure Service Bus به طور محکم با اکوسیستم ادغام می‌شود. برای اتصال بی‌دردسر به سرویس‌های AWS .NET، همه پلتفرم‌ها SDK‌های قوی ارائه می‌دهند.

## مقیاس‌پذیری و عملکرد: مدیریت حرارت

Apache Kafka در مورد مقیاس‌پذیری، قهرمان سنگین وزن است، طراحی شده برای مدیریت جریان‌های عظیم داده در خوشه‌های توزیع شده از RabbitMQ. از مقیاس‌پذیری افقی از طریق خوشه‌بندی و فدراسیون پشتیبانی می‌کند. Azure Service Bus قابلیت‌های مقیاس‌پذیری خودکار در ابر ساده می‌کند، در حالی که Amazon SQS از توان عملیاتی تقریباً نامحدود برخوردار است.

## تصمیم‌های تحويل: رساندن پیام

صفات پیام سنگ بنای ارتباط قابل اعتماد هستند، و تصمیم‌های تحويلی که ارائه می‌دهند - حداقل یک بار، حداقل یک بار، یا دقیقاً یک بار - می‌توانند متفاوت باشد. RabbitMQ اعطاف‌پذیری را با حالت‌های تأیید ارائه می‌دهد. Apache Kafka به طور پیش‌فرض تحويل حداقل یک بار را ارائه می‌دهد. Azure Service Bus پشتیبانی بومی از تحويل دقیقاً یک بار در پیام‌رسانی تراکنشی ارائه می‌دهد.

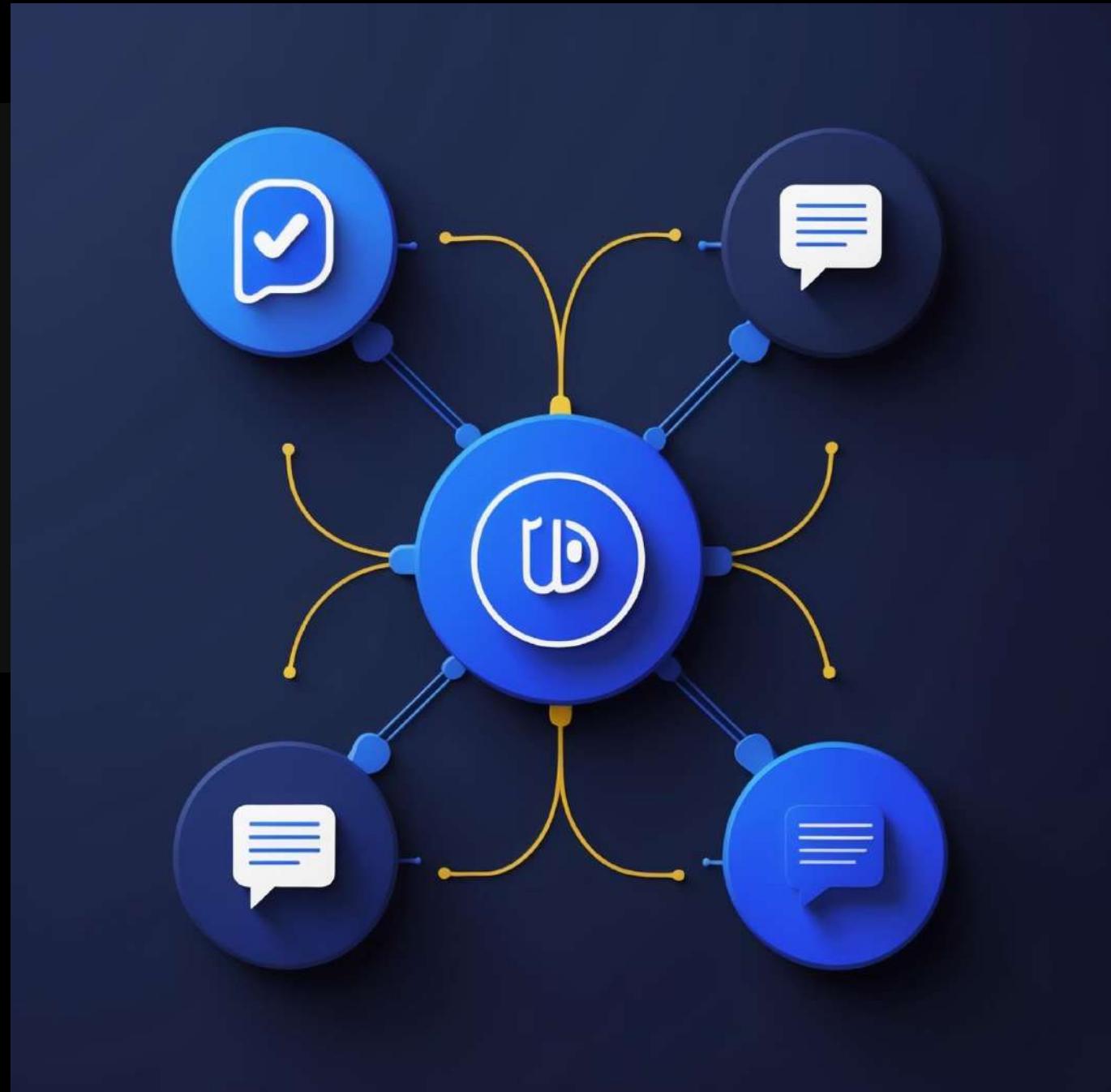
انتخاب صفات پیام مناسب به یافتن تعادل درست ویژگی‌ها برای نیازهای خاص شما بستگی دارد. اکوسیستم .NET اطمینان می‌دهد که می‌توانید انتخاب خود را با اطمینان و دقت پیاده‌سازی کنید.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# #C در پیام سازی صفحه

در حالی که WebHook‌ها در ارائه اعلان‌های بلادرنگ از یک سرویس به سرویس دیگر برتری دارند، آنها به در دسترس بودن هر دو فرستنده و گیرنده در لحظه دقیق وقوع رویداد متکی هستند. اما چه اتفاقی می‌افتد وقتی گیرنده به طور موقت آفلاین است یا با سیلی از رویدادهای ورودی روبرو می‌شود؟ اینجاست که صفحه‌ای پیام به عنوان قهرمانان ناشناخته ارتباط ناهمگام عمل می‌کنند.



<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

# راهاندازی محیط

برای شروع کار با صفحه‌ای پیام در.NET، به یک محیط پیکربندی شده مناسب نیاز دارید تا بتوانید پیام‌ها را به طور یکپارچه تولید و مصرف کنید. در این مثال، ما بر RabbitMQ به عنوان ارائه‌دهنده صفت خود تمرکز می‌کنیم.

```
docker run -d --hostname rabbitmq-host --name rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:3-management
```

این دستور RabbitMQ را با رابط مدیریتی در آدرس <http://localhost:15672> راهاندازی می‌کند. از اعتبارنامه‌های پیش‌فرض (guest/guest) برای ورود استفاده کنید. حالا، بباید راه حل.NET را راهاندازی کنیم.

```
dotnet new console -n MessageQueueDemo cd MessageQueueDemo dotnet add package RabbitMQ.Client
```

سپس، تولیدکننده پیام را مقداردهی اولیه کنید. در فایل Program.cs خود، با ایجاد یک اتصال به RabbitMQ و ارسال یک پیام آزمایشی شروع کنید:

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# کد نمونه برای تولیدکننده پیام

```
using RabbitMQ.Client;using System.Text;var factory = new ConnectionFactory() { HostName = "localhost" };using var connection =  
factory.CreateConnection();using var channel = connection.CreateModel();channel.QueueDeclare(queue: "demo-queue", durable: false,  
exclusive: false, autoDelete: false, arguments: null);string message = "Hello, RabbitMQ!";var body =  
Encoding.UTF8.GetBytes(message);channel.BasicPublish(exchange: "", routingKey: "demo-queue", basicProperties:  
null, body: body);Console.WriteLine($"[x] Sent '{message}'");
```

# کد نمونه برای مصرف‌کننده پیام

```
using RabbitMQ.Client;using RabbitMQ.Client.Events;using System.Text;var factory = new ConnectionFactory() { HostName = "localhost" };using var connection =  
factory.CreateConnection();using var channel = connection.CreateModel();channel.QueueDeclare(queue: "demo-queue", durable: false, exclusive:  
false, autoDelete: false, arguments: null);var consumer = new EventingBasicConsumer(channel);consumer.Received += (model, ea) =>{ var body  
= ea.Body.ToArray(); var message = Encoding.UTF8.GetString(body); Console.WriteLine($"[x] Received '{message}'");};channel.BasicConsume(queue: "demo-queue",  
autoAck: true, consumer: consumer);Console.WriteLine("Press [enter] to exit.");Console.ReadLine();
```

<https://www.nuget.org/packages/RabbitMQ.Client>

این مصرف‌کننده به پیام‌ها در demo-queue گوش می‌دهد و آنها را در کنسول چاپ می‌کند. هر دو برنامه را همزمان اجرا کنید: تولیدکننده پیام‌ها را ارسال می‌کند و مصرف‌کننده آنها را دریافت می‌کند.

<https://github.com/MohamadHoseinRoohiAmini>

# ایجاد یک تولیدکننده پیام

با راهاندازی محیط شما، گام بعدی طراحی یک مولفه مسئول ارسال پیامها به صفر است. این شامل ایجاد یک تولیدکننده است که نه تنها یک اتصال به صفر پیام برقرار می‌کند و پیام‌ها را آماده می‌کند، بلکه نقش مهمی در تضمین ارتباط قابل اعتماد در سیستم توزیع شده شما دارد.

```
using RabbitMQ.Client;using System.Text;public class MessageProducer{ private readonly string _queueName; private readonly IConnection _connection; private readonly IModel _channel; public MessageProducer(string queueName, string hostName = "localhost") { _queueName = queueName; var factory = new ConnectionFactory { HostName = hostName }; _connection = factory.CreateConnection(); _channel = _connection.CreateModel(); _channel.QueueDeclare( queue: _queueName, durable: false, exclusive: false, autoDelete: false, arguments: null ); } public void SendMessage(string message) { var body = Encoding.UTF8.GetBytes(message); _channel.BasicPublish(exchange: "", routingKey: _queueName, basicProperties: null, body: body); Console.WriteLine($"[x] Sent: {message}"); } public void Dispose() { _channel.Close(); _connection.Close(); }}
```

# استفاده از کلاس تولیدکننده پیام

این کلاس راهاندازی اتصال را کپسوله می‌کند و متده ارسال پیامها ارائه می‌دهد. متده SendMessage یک رشته را می‌گیرد، آن را به بایت تبدیل می‌کند و در صف مشخص شده در سازنده منتشر می‌کند.

سپس، از این کلاس در برنامه خود برای تولید پیامها استفاده کنید Program.cs. خود را به روزرسانی کنید تا شامل موارد زیر باشد:

```
using System; class Program{    static void Main()    {        const string queueName = "demo-queue";        using var producer = new MessageProducer(queueName);        while (true)        {            Console.Write("Enter a message to send (or type 'exit' to quit): ");            var message = Console.ReadLine();            if (string.IsNullOrEmpty(message) || message.ToLower() == "exit")                break;            producer.SendMessage(message);        }        Console.WriteLine("Producer terminated.");    }}
```

# ساخت یک مصرف‌کننده پیام

با وجود یک تولیدکننده برای ارسال پیام‌ها به صفر، گام بعدی ساخت همتایی است که این پیام‌ها را بازیابی و پردازش می‌کند. این مؤلفه برای تبدیل داده‌های در صفر به نتایج قابل اجرا در برنامه شما بسیار مهم است.

```
using RabbitMQ.Client;using RabbitMQ.Client.Events;using System.Text;public class MessageConsumer{ private readonly string _queueName; private readonly IConnection _connection; private readonly IModel _channel; public MessageConsumer(string queueName, string hostName = "localhost") { _queueName = queueName; var factory = new ConnectionFactory { HostName = hostName }; _connection = factory.CreateConnection(); _channel = _connection.CreateModel(); _channel.QueueDeclare( queue: _queueName, durable: false, exclusive: false, autoDelete: false, arguments: null ); } public void StartListening() { var consumer = new EventingBasicConsumer(_channel); consumer.Received += (model, eventArgs) => { var body = eventArgs.Body.ToArray(); var message = Encoding.UTF8.GetString(body); Console.WriteLine($"[x] Received: {message}"); // شبیه‌سازی پردازش پیام ProcessMessage(message); }; _channel.BasicConsume( queue: _queueName, autoAck: true, consumer: consumer ); Console.WriteLine("Listening for messages. Press [Enter] to exit."); Console.ReadLine(); } private void ProcessMessage(string message) { Console.WriteLine($"[✓] Processed: {message}"); } public void Dispose() { _channel.Close(); _connection.Close(); }}
```

# مدیریت تأییدها و خطاها

پردازش قابل اعتماد پیام‌ها به بیش از بازیابی و عمل بر روی آنها نیاز دارد؛ همچنین شامل مدیریت تأییدها و مدیریت مؤثر خطاها است. مدیریت مناسب تأیید می‌تواند از گم شدن پیام‌های مهم یا پردازش چندباره آنها جلوگیری کند.

2

## مدیریت خطا

سپس، رویداد Received را تغییر دهید تا پس از پردازش موفق یک پیام، یک تأیید ارسال کند:

```
consumer.Received += (model, eventArgs) =>{ var body = eventArgs.Body.ToArray(); var message = Encoding.UTF8.GetString(body); Console.WriteLine($"[x] Received: {message}"); try { ProcessMessage(message); // _channel.BasicAck(deliveryTag: eventArgs.DeliveryTag, multiple: false); Console.WriteLine($"[✓] Acknowledged: {message}"); } catch (Exception ex) { Console.WriteLine($"[!] Error processing message: {ex.Message}"); // دادن مجدد در صورت برای تلاش دیگر _channel.BasicNack(deliveryTag: eventArgs.DeliveryTag, multiple: false, requeue: true); }}
```

1

## تأیید دستی

به طور پیش‌فرض، برخی صفاتی پیام، مانند RabbitMQ، از حالت تأیید خودکار استفاده می‌کنند که در آن پیام‌ها به محض تحويل به مصرف‌کننده به عنوان پردازش شده علامت‌گذاری می‌شوند. برای تغییر به تأیید دستی، متدهای BasicConsume و MessageConsumer را در کلاس MessageConsumer خود به روزرسانی کنید:

```
_channel.BasicConsume(queue: _queueName, autoAck: false, consumer: consumer);
```

# صف نامه‌های مرده (Dead Letter Queue)

برای جلوگیری از پردازش مجدد بی‌پایان پیام‌های مشکل‌دار، سیستم ما مجهز به یک صف نامه‌های مرده (DLQ) برای مدیریت پیام‌های مشکل‌دار است. صف‌های نامه‌های مرده، یک ویژگی کلیدی سیستم ما، پیام‌هایی را که از حد تلاش مجدد فراتر می‌روند یا با خطاهای غیرقابل بازیابی مواجه می‌شوند، جمع‌آوری می‌کنند و اجازه می‌دهند به طور جداگانه تحلیل شوند.

```
_channel.QueueDeclare(    queue: _queueName,    durable: false,    exclusive: false,    autoDelete: false,    arguments: new Dictionary<string, object> {        "x-dead-letter-exchange": "dead-letter-exchange",        "x-dead-letter-routing-key": "dead-letter-queue"    })=_channel.QueueDeclare(    queue: "dead-letter-queue",    durable: false,    exclusive: false,    autoDelete: false,    arguments: null);
```

اکنون، هر پیامی که چندین بار با شکست مواجه شود، به طور خودکار به dead-letter-queue هدایت می‌شود، یک صف ویژه که در آن چنین پیام‌هایی برای تحلیل بیشتر ذخیره می‌شوند، از اختلال بیشتر در صف اصلی شما جلوگیری می‌کند.

<https://www.linkedin.com/in/mhramini/>

با پیاده‌سازی تأییدهای دستی، صف‌های نامه‌های مرده و ثبت خطا، اطمینان حاصل می‌کنید که صف پیام شما می‌تواند با شکست‌ها به طور مناسب برخورد کند و در عین حال یکپارچگی داده را حفظ کند.

<https://github.com/MohamadHoseinKoohiAmini>

# موضوعات پیشرفتی در صفحه پیام

## بهینه‌سازی عملکرد و مقیاس‌پذیری

ساخت یک سیستم صفحه پیام با عملکرد بالا و مقیاس‌پذیر فقط یک وظیفه فنی نیست، بلکه یک وظیفه حیاتی است. این نیازمند بهینه‌سازی توان عملیاتی پیام، کاهش تأخیر و اطمینان از مقیاس‌پذیری بی‌درنگ سیستم با رشد تقاضا است.

## امنیت صفحه‌های پیام

امنیت یک ملاحظه مهم در طراحی یک سیستم صفحه پیام است. تخصص شما در پیاده‌سازی مکانیسم‌های احراز هویت، رمزگذاری و کنترل دسترسی بسیار مهم است. بدون این محافظت‌ها، پیام‌های حساس می‌توانند رهگیری، دستکاری یا سوء استفاده شوند.



## تضمين‌های تحويل پیام

تضمين‌های تحويل قابل اعتماد یکی از ستون‌های اصلی ساخت سیستم‌های پیام‌رسانی قوی است. مدل تضمين تحويل را تعریف کنید: حداکثر یک بار، حداقل یک بار، یا دقیقاً یک بار. هر کدام مبادلاتی دارد و انتخاب درست به نیازهای خاص برنامه شما بستگی دارد.

## بهینه‌سازی عملکرد و بهترین شیوه‌ها

همانطور که صفحه‌های پیام در معماری توزیع شده شما نقش اصلی را ایفا می‌کنند، کارایی اهمیت پیدا می‌کند. در حالی که یک راهاندازی پایه ممکن است برای حجم کاری کوچک کافی باشد، مقیاس‌پذیری به سیستم‌های سطح سازمانی یا مدیریت سناریوهای با توان عملیاتی بالا نیازمند بهینه‌سازی است.

با پیاده‌سازی این تکنیک‌های پیشرفتی، سیستم صفحه پیام شما بارهای سنگین را به طور کارآمد مدیریت می‌کند و در عین حال مقیاس‌پذیری و قابلیت اطمینان را حفظ می‌کند. این تکنیک‌ها اطمینان می‌دهند که سیستم شما برای نیازهای سطح تولید آماده است و آماده انطباق با رشد آینده است.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# نتیجه‌گیری

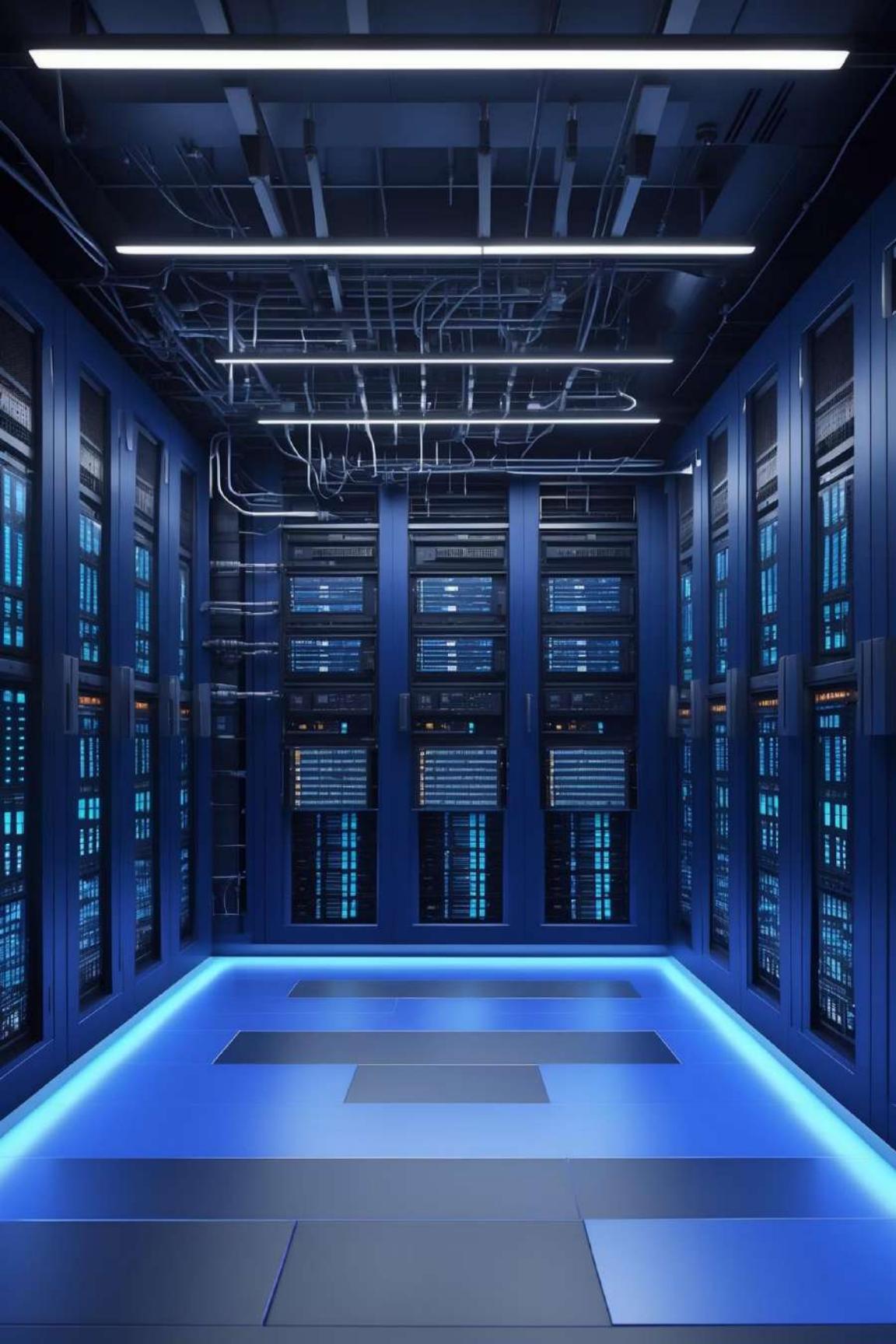
صفهای پیام ستون فقرات برنامه‌های مدرن و توزیع شده را تشکیل می‌دهند و راه حلی قوی برای مسائل رایج در طراحی سیستم‌های مقیاس‌پذیر و انعطاف‌پذیر ارائه می‌دهند. همانطور که در این راهنمای بررسی شد، از مفاهیم اصلی مانند عدم وابستگی و ارتباط ناهمزنان گرفته تا پیاده‌سازی‌های عملی با RabbitMQ در C#، درک و به کارگیری صفحه‌ای پیام برای هر توسعه‌دهنده‌ای که به دنبال ساخت سیستم‌های قابل اعتماد و با عملکرد بالا است، ضروری است.

با پذیرش تأییدهای دستی، صفحه‌ای نامه‌های مرده، و در نظر گرفتن موضوعات پیشرفتی مانند تضمین‌های تحويل و امنیت، می‌توانید سیستم‌های پیام‌رسانی بسازید که نه تنها کارآمد هستند بلکه در برابر شکست‌ها نیز مقاوم هستند. این فناوری‌ها به شما قدرت می‌دهند تا معماری‌هایی را طراحی کنید که می‌توانند بارهای کاری سنگین را مدیریت کنند، ارتباطات بین سرویس‌ها را تسهیل کنند و تجربه کاربری یکپارچه‌ای را ارائه دهند. با ابزارهایی که اکنون در اختیار دارید، آماده‌اید تا قدرت کامل صفحه‌ای پیام را برای حل چالش‌های پیچیده برنامه‌نویسی به کار بگیرید.

# SignalR در دنیای برنامه‌نویسی شبکه

ارتباطات بلاذرنگ در برنامه‌نویسی مدرن شبکه

<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>



# فهرست مطالب

## پیاده‌سازی سرور

راهاندازی و پیکربندی سرور SignalR

## مفاهیم اصلی

هاب‌ها، اتصالات و پروتکل‌ها

## معرفی SignalR

آشنایی با ارتباطات بلادرنگ و مزایای SignalR

## ویژگی‌های پیشرفته

گروه‌بندی، مقیاس‌پذیری و اشکال‌زدایی

## کلاینت‌ها

ایجاد کلاینت‌های SignalR در پلتفرم‌های مختلف

# معرفی SignalR: ارتباطات بلاذرنگ

در دنیای برنامه‌نویسی مدرن شبکه، نیاز به ارتباطات بلاذرنگ به یکی از ارکان اصلی بسیاری از برنامه‌ها تبدیل شده است. از سیستم‌های گفتگوی زنده تا ابزارهای ویرایش مشارکتی، کاربران به طور فزاینده‌ای انتظار دارند که به روزرسانی‌ها و تعاملات فوراً و بدون نیاز به بارگذاری مجدد صفحه یا مکانیزم‌های پیچیده نظرسنجی رخ دهد.

پیچیدگی‌های ارتباطات بلاذرنگ را ساده می‌کند و به .NET 8، یک کتابخانه قدرتمند در SignalR با ایجاد پلی بین توسعه‌دهندگان امکان می‌دهد تجربه‌های کاربری بی‌درنگ و پاسخگو ایجاد کنند. نه تنها انعطاف‌پذیری و کارایی را SignalR سرور و کلاینت از طریق یک کانال ارتباطی دوطرفه و پویا، در پروتکل‌های مختلف انتقال مختلف ارائه می‌دهد، بلکه قدرت ایجاد یک تجربه کاربری عالی را در اختیار شما قرار می‌دهد.



<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

# بلاذرنگ، همه وقت: معرفی SignalR

## انطباق‌پذیری

برای بهترین WebSockets از SignalR عملکرد استفاده می‌کند، اما به طور خودکار به پروتکل‌های جایگزین مانند Server-Sent Events و Long Polling برمی‌گردد، در دسترس WebSockets در صورتی که نباشد.

## تغییر پارادایم

تغییری اساسی در نحوه تفکر ما SignalR درباره پاسخگویی برنامه‌ها ایجاد می‌کند. با ایجاد اتصالات پایدار بین کلاینت‌ها و سرورها، امکان ویژگی‌هایی را فراهم می‌کند که قبلًا فقط در سیستم‌های بسیار پیچیده و منابع بر امکان‌پذیر بود.

## فراتر از WebHook‌ها

برخلاف WebHook‌ها که در فصل قبل بررسی شدند، SignalR ارتباط دوطرفه و مدام را برای برنامه‌های بلاذرنگ فراهم می‌کند.

امکان ایجاد برنامه‌هایی با قابلیت‌های بلاذرنگ مانند امتیازات ورزشی زنده، ابزارهای ویرایش مشارکتی و داشبوردهای بلاذرنگ را به شکلی ساده و کارآمد فراهم می‌کند.

# ضربان بلاذرنگ : درک فضای مسئله

جذابیت برنامه‌های بلاذرنگ در فوریت آنها نهفته است-داده‌ها همزمان با وقوع رویدادها ظاهر می‌شوند، تعاملات فوری احساس می‌شوند و کاربران در یک تجربه پویا و پاسخگو غرق می‌شوند. با این حال، دستیابی به این سطح از پاسخگویی چالش‌های منحصر به فردی را ارائه می‌دهد.

مدل‌های سنتی درخواست-پاسخ، اگرچه قابل اعتماد هستند، تأخیر و ناکارآمدی را برای سناریوهای بلاذرنگ معرفی می‌کنند. نظرسنجی مکرر یا بارگذاری‌های دستی، هم سرورها و هم کلاینت‌ها را تحت فشار قرار می‌دهد و گلوگاهی ایجاد می‌کند که تجربه بی‌وقفه‌ای را که کاربران می‌خواهند، تضعیف می‌کند.

در هسته این چالش، حفظ ارتباط مداوم بین کلاینت و سرور قرار دارد. بدون مکانیزمی برای ارسال به روزرسانی‌ها توسط سرورها، برنامه‌ها نسبت به تغییرات خارج از اقدامات کاربر نابینا می‌مانند. به عنوان مثال، یک برنامه معاملات سهام نیاز دارد به روزرسانی‌های بازار را در زمان واقعی نشان دهد، نه زمانی که کاربر روی "تازه‌سازی" کلیک می‌کند.

این مشکل را با یک انتزاع توسعه‌دهنده‌پسند بر روی SignalR WebSockets کتابخانه و سایر پروتکل‌های انتقال حل می‌کند. پیچیدگی‌های مدیریت اتصال، مذاکره پروتکل و مقیاس‌پذیری را خودکار می‌کند، بنابراین می‌توانید بر ارائه ویژگی‌های بلاذرنگ تمرکز کنید.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# آنچه SignalR را درخشن می‌کند: ویژگی‌ها و معماری کلیدی

نمای کلی معماری SignalR

## گروهبندی اتصال

از گروهبندی اتصال پشتیبانی SignalR می‌کند و کنترل دقیق بر روی اینکه چه کسی به روزرسانی‌ها را دریافت می‌کند، فراهم می‌کند، خواه پخش به همه کلاینت‌های متصل یا هدف‌گیری کاربران یا گروه‌های خاص.

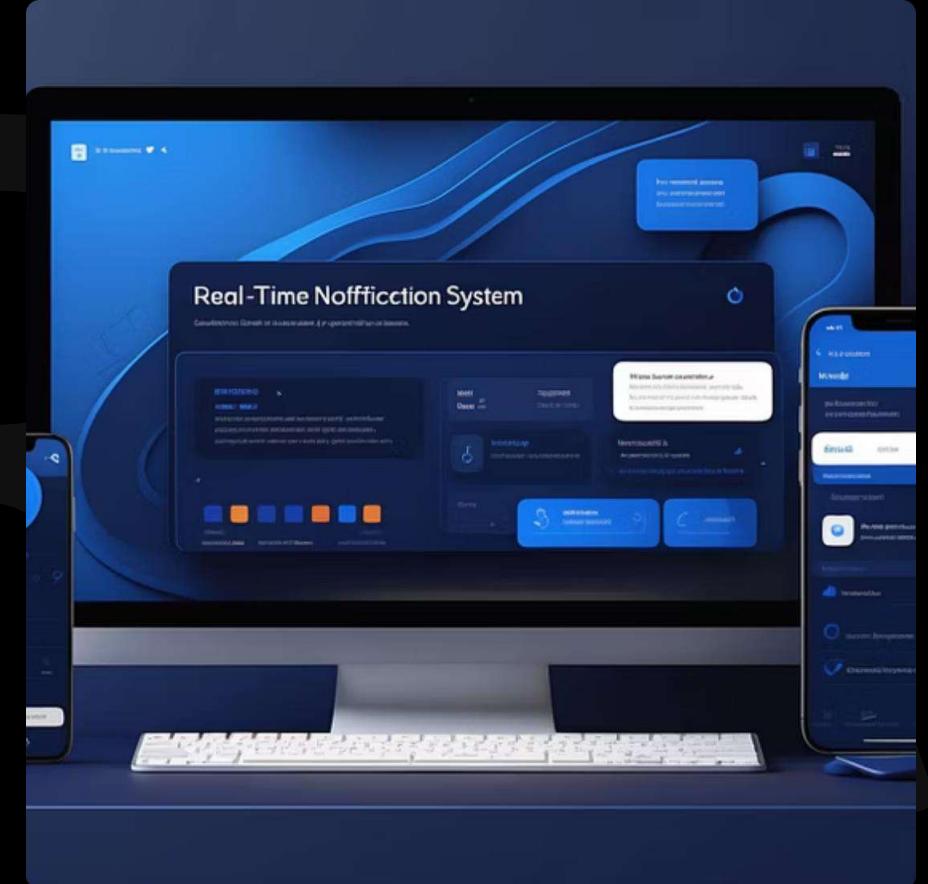
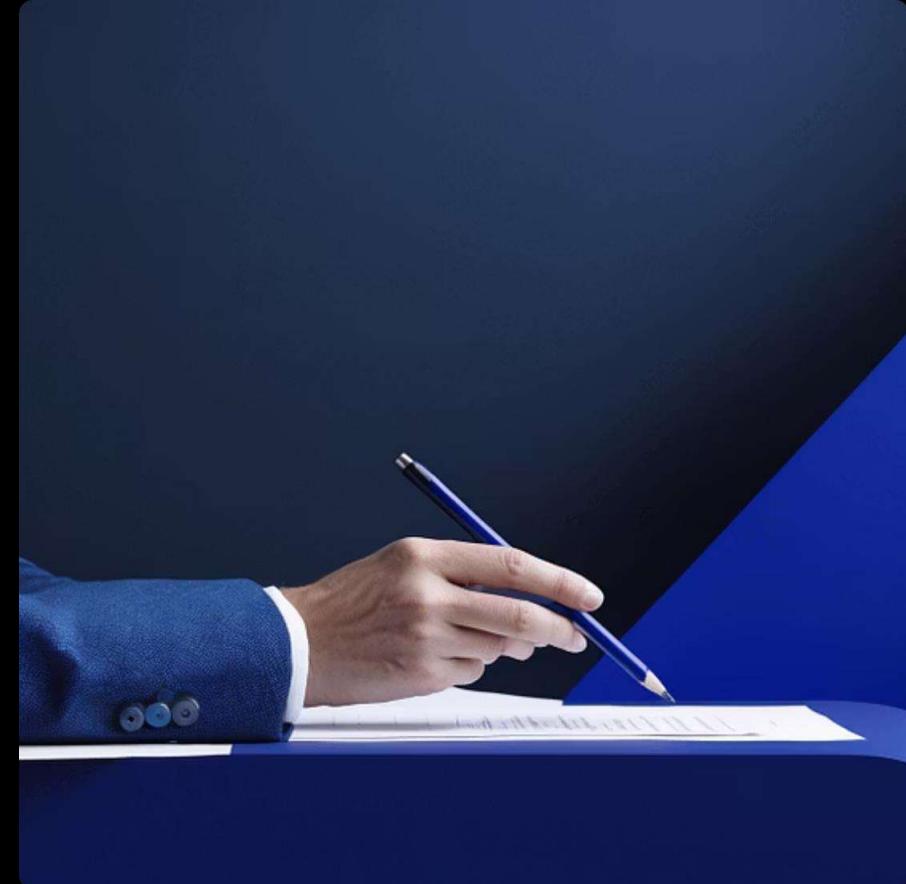
## اتصالات پایدار

یکی از ویژگی‌های برجسته SignalR توانایی آن در مدیریت اتصالات پایدار است، که تفاوت کلیدی با مدل‌های سنتی درخواست-پاسخ HTTP است. این اتصالات پایدار ارتباط دوطرفه را فعال می‌کنند و به سرورها اجازه می‌دهند به روزرسانی‌های کلاینت را بدون انتظار برای درخواست ارسال کنند.

## انتزاع پروتکل‌های انتقال

انتزاعی بر روی چندین پروتکل WebSockets، Server-Sent Events (SSE) و Long Polling انتقال از جمله این مکانیزم بازگشت هوشمند. می‌کند اطمینان می‌دهد که برنامه‌های شما تجربه بلادرنگ بی‌نقصی را ارائه می‌دهند، صرف نظر از محیط کلاینت یا قابلیت‌های مرورگر.

# مزیت SignalR: کاربردها و سناریوهای دنیای واقعی



## داشبوردهای بلادرنگ

خواه پیگیری داده‌های بازار سهام، نظارت بر سلامت سرور، یا تجسم معیارهای سنسور IoT، به شما امکان می‌دهد بهروزرسانی‌ها را مستقیماً به داشبورد بدون دخالت کاربر ارسال کنید. ارتباط کم تأخیر آن اطمینان می‌دهد که این معیارها دقیق و بهروز باقی می‌مانند.

تصویر کنید چندین کاربر هم‌زمان یک سند را ویرایش می‌کنند و تغییرات را در زمان واقعی می‌بینند. پلتفرم‌هایی مانند ابزارهای اشتراک کد، برنامه‌های تخته سفید و سیستم‌های گفتگوی خدمات مشتری به شدت به توانایی SignalR برای همگام‌سازی فوری بهروزرسانی‌ها بین کلاینت‌ها متکی هستند.

## ابزارهای مشارکتی

ارائه بهروزرسانی‌های فوری برای فیدهای رسانه‌های اجتماعی، اطلاع‌رسانی به کاربران درباره تغییرات در محیط‌های مشارکتی، یا ارائه هشدارهای وضعیت در برنامه‌های سازمانی. با SignalR، این اعلان‌ها بدون زحمت می‌شوند.

## اعلان‌های زنده

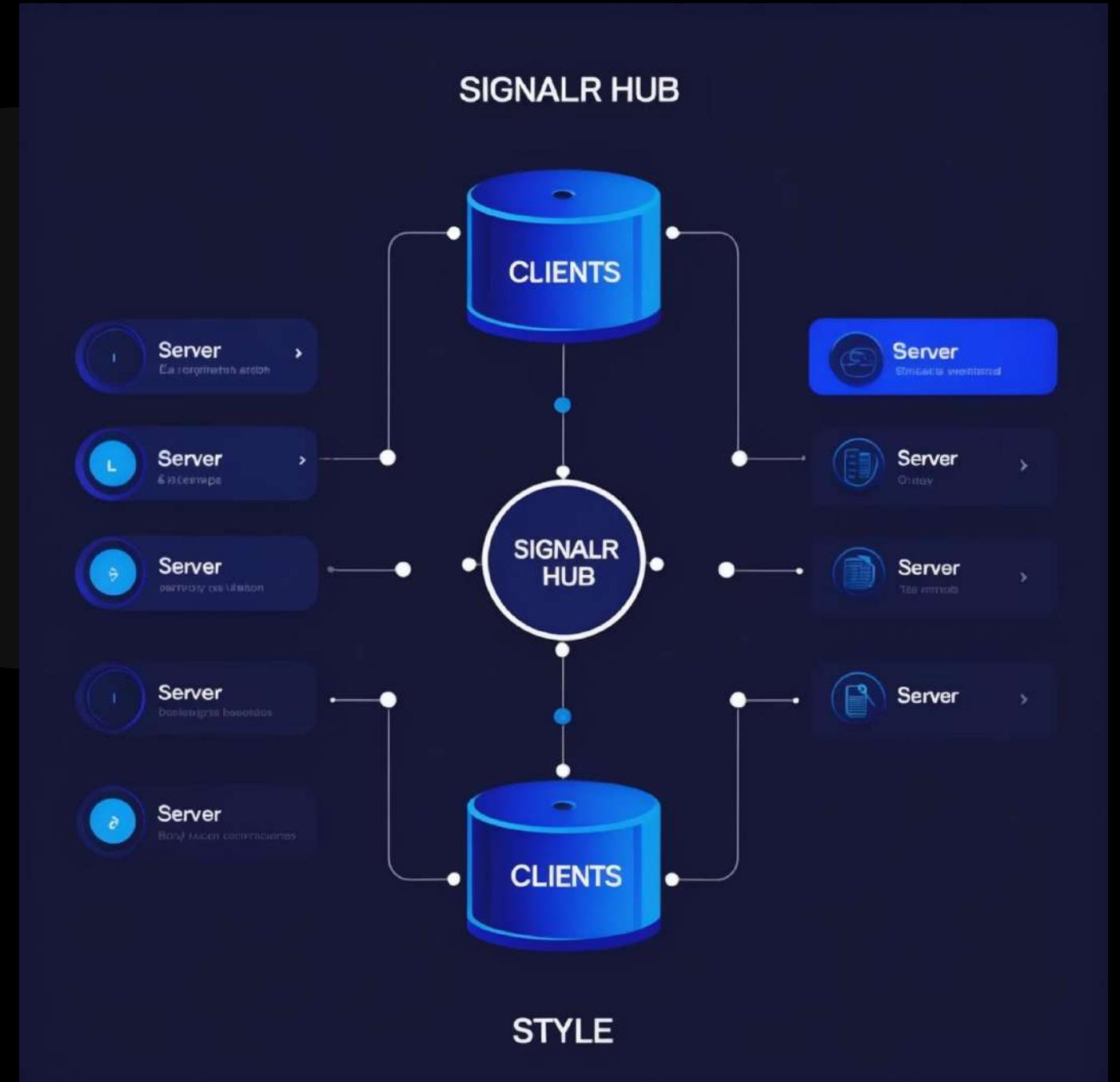
# جعبه ابزار SignalR: هابها، اتصالات و پروتکل‌ها

## هاب‌ها: رهبران ارکستر

هاب‌ها هسته اصلی SignalR هستند که ارتباط بین کلاینت‌ها و سرورها را هماهنگ می‌کنند. آنها فرآیند فراخوانی متدهای سمت سرور از کلاینت‌ها (و بر عکس (را با انتزاع پیچیدگی‌های پروتکل‌های زیربنایی ساده می‌کنند. هاب‌ها به شما امکان می‌دهند بر منطق برنامه خود تمرکز کنید، نه بر جزئیات انتقال داده.

## اتصالات پایدار

اتصالات پایدار را مدیریت می‌کند، اتصال مجدد را به طور مناسب انجام می‌دهد و به طور پویا SignalR این تغییر می‌دهد WebSockets و Long Polling و Server-Sent Events بین پروتکل‌هایی مانند اطمینان می‌دهد که برنامه شما یک تجربه بلاذرنگ قوی را صرف نظر از سناریو ارائه می‌دهد.



<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

# قلب هاب : مدیریت ارتباطات بلاذرنگ



## گروه‌بندی و شخصی‌سازی

هاب بیش از یک مسیریاب است - یک پلتفرم برای شخصی‌سازی و مقیاس‌پذیری است. به توسعه‌دهندگان امکان می‌دهد کلاینت‌ها را در گروه‌ها سازماندهی کنند و ارتباطات هدفمند را فعال کنند - تصور کنید پخش به اتاق‌های خاص در یک برنامه گفتگو یا ارسال اعلان‌ها فقط به کاربرانی که یک موضوع خاص را دنبال می‌کنند.



## مدیریت اتصال

به طور بی‌درنگ اتصالات را مدیریت SignalR می‌کند، اتصالات فعال را به طور خودکار رديابی می‌کند، قطع اتصال‌ها را به خوبی مدیریت می‌کند و از منطق اتصال مجدد زمانی که یک کلاینت به طور موقت آفلاین است، پشتیبانی همه اینها به یک تجربه ارتباطی . می‌کند بلاذرنگ قابل اعتماد و لذت‌بخش کمک می‌کند.



## ارتباط دوطرفه

کلاینت‌ها می‌توانند متدهایی را در سرور فراخوانی کنند، در حالی که سرور می‌تواند پیام‌ها را به یک، چند یا همه کلاینت‌های متصل پخش کند. در یک برنامه گفتگوی زنده، یک کلاینت می‌تواند پیامی را از طریق هاب به سرور ارسال کند و آن را به سایر کاربران متصل در زمان واقعی منتقل کند.

# ساخت ستون فقرات : راهاندازی سرور SignalR شما

در هسته هر برنامه بلادرنگ، یک سرور SignalR قرار دارد - ستون فقراتی که اتصالات را مدیریت می کند، پیامها را مسیریابی می کند و ارتباط بی درنگ کلاینت را تضمین می کند . راهاندازی این سرور فقط یک کار فنی نیست؛ اولین قدم به سوی ایجاد برنامه هایی است که زنده، فوراً پاسخگو و بی نهایت جذاب احساس می شوند.

در این بخش، شما را در پیکربندی سرور SignalR خود در .NET راهنمایی می کنیم، از نصب وابستگی ها تا تعریف هابها و نگاشت نقاط پایانی . این کار، اگرچه چالش برانگیز است، بسیار پاداش دهنده است . در طول مسیر، خواهید دید که چگونه سرور خود را برای تطبیق با نیازهای منحصر به فرد برنامه خود سفارشی کنید.



<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

# پایه‌گذاری: نصب و پیکربندی SignalR

## پیکربندی سرور

```
using SignalRServerExample;var  
builder =  
WebApplication.CreateBuilder(args);builder.Services.AddSignalR();var app  
=builder.Build();app.UseHttpsRedirection();app.UseRouting();app.MapHub("/  
chathub");app.Run();
```

## ایجاد هاب

```
using  
Microsoft.AspNetCore.SignalR;public  
class ChatHub : Hub{    public async  
Task SendMessage(string user, string  
message) {        await  
Clients.All.SendAsync("ReceiveMessag  
e", user, message);    }}
```

## راهاندازی پروژه

```
dotnet new web -n  
SignalRServerExamplecd  
SignalRServerExampledotnet add  
package Microsoft.AspNetCore.SignalR
```

ابتدا یک پروژه ASP.NET Core جدید ایجاد کنید و بسته NuGet SignalR را به راه حل خود اضافه کنید.

میان افزار SignalR را در برنامه ASP.NET Core خود پیکربندی کنید. متدهای MapHub و ChatHub را به نقطه پایانی /chathub نگاشت می‌کند و به کلاینت‌ها اجازه اتصال می‌دهد.

هاب قلب ارتباطات SignalR است، یک کلاس که تعامل بین کلاینت‌ها و سرور را تسهیل می‌کند.

# نتیجه‌گیری

همانطور که در این مرور جامع دیدید، SignalR بیش از یک ابزار است؛ آن یک کاتالیزور برای ایجاد تجربه‌های وب پویا و بلادرنگ است. با انتزاع پیچیدگی‌های پروتکل‌های ارتباطی و ارائه یک API ساده و قدرتمند مبتنی بر هاب، توسعه‌دهندگان را قادر می‌سازد تا بدون غرق شدن در جزئیات فنی زیربنایی، بر ساخت برنامه‌های نوآورانه تمرکز کنند.

از اعلان‌های زنده و ابزارهای مشارکتی گرفته تا داشبوردهای بلادرنگ و بازی‌ها، SignalR دروازه‌های بی‌شماری را به روی ارتباطات یکپارچه و کم‌تأخیر باز می‌کند. با درک هاب‌ها، مدیریت اتصالات و پیکربندی سرور، شما اکنون مجهز به ایجاد برنامه‌هایی هستید که نه تنها کارآمد هستند، بلکه برای کاربران نیز جذاب و واکنش‌گرا هستند.

با SignalR، دیگر نیازی به تکیه بر مدل‌های درخواست-پاسخ سنتی نیست. در عوض، می‌توانید آینده وب را در آغوش بگیرید، جایی که سرور می‌تواند به طور فعال به روزرسانی‌ها را به کلاینت‌ها ارسال کند و تجربه‌ای واقعاً بلادرنگ را برای همه فراهم کند.



# نگاهی به آینده با QUIC

پروتکل‌های شبکه همیشه برای حل مشکلات جدید و استفاده از فرصت‌های تازه تغییر کرده‌اند) QUIC. اتصالات اینترنتی سریع (UDP یک پروتکل جدید است که برای حل مشکلات پروتکل‌های قدیمی‌تر مثل TCP ساخته شده است. این پروتکل که توسط گوگل ایجاد شده و حالا یک استاندارد جهانی است، می‌خواهد با بهتر کردن سرعت و امنیت، تجربه استفاده از اینترنت را خیلی بهتر کند.

امروزه که بیشتر از همیشه به اینترنت نیاز داریم، ارتباطات سریع‌تر، پایدارتر و امن‌تر بسیار مهم هستند. پروتکل‌های قدیمی‌تر، با اینکه خوب هستند، در شبکه‌های جدید، به خصوص در موبایل و شبکه‌های بی‌سیم، مشکلاتی مثل کندی، کاهش سرعت و امنیت کم دارند. QUIC با راه‌های جدیدی مثل شروع سریع اتصال، مدیریت موازی اطلاعات (Stream) و رمزگاری پیش‌فرض، می‌خواهد این مشکلات را برطرف کند و راه را برای برنامه‌های اینترنتی آینده باز کند.

# چیست؟ یک پروتکل مدرن برای نیازهای مدرن QUIC

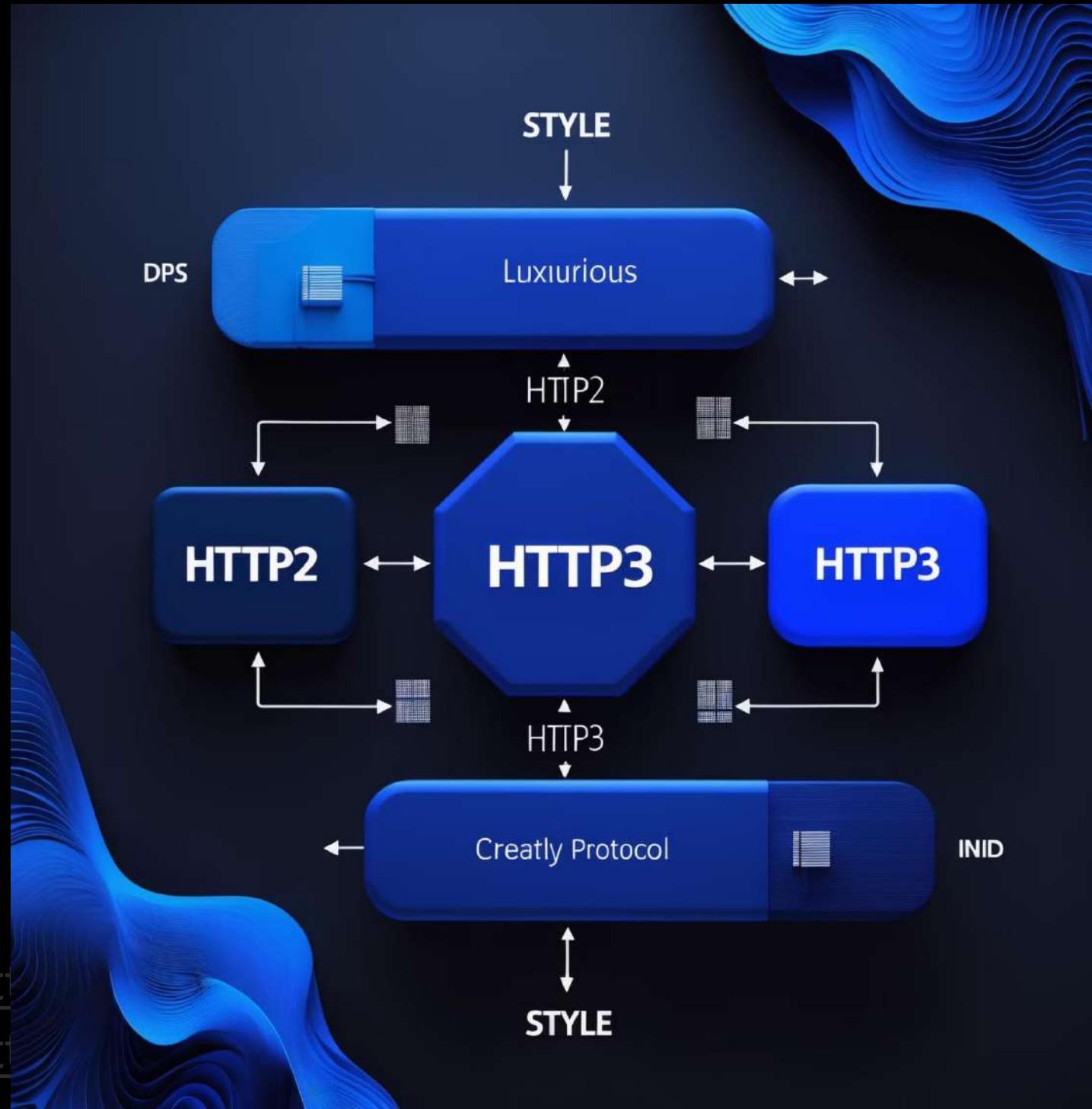


<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>

# تفاوت‌های کلیدی HTTP/2 در مقابل HTTP/3

درک تفاوت‌های اصلی این دو نسخه برای کسانی که می‌خواهند سرعت وب را بهتر کنند یا این تغییر اساسی، مزایای مهمی در سرعت و امنیت دارد که برای شبکه‌های امروزی حیاتی است. استفاده می‌کند TCP QUIC بود، از 2/HTTP که اساس این پروتکل، به جای یک قدم بزرگ رو به جلو در پروتکلهای وب است 3/HTTP برنامه‌های جدید بسازند، ضروری است.



# پایه UDP با بجهودهای مدرن



## انعطاف‌پذیری و نوآوری

یکی از بزرگترین خوبی‌های QUIC، انعطاف‌پذیری آن است. چون قابلیت اطمینان و امنیت را خودش (بالاتر از UDP) مدیریت می‌کند، QUIC می‌تواند بدون نیاز به بهروزرسانی سیستم عامل، سریعاً پیشرفت کند. این به برنامه‌نویسان اجازه می‌دهد تا امکانات جدید را اضافه کرده و پروتکل را با نیازهای آینده شبکه و فناوری‌ها هماهنگ کنند. این انعطاف‌پذیری باعث می‌شود QUIC برخلاف TCP که در برابر تغییرات کند است، بهتر عمل کند.

## پایه UDP و قابلیت اطمینان

استفاده از UDP پروتکل ساده و بدون نیاز به اتصال QUIC با این حال، دوری کند تا از مشکلات با (خودش ویژگی‌های مهمی مثل اطمینان از رسیدن داده‌ها، کنترل سرعت داده‌ها (استفاده از شماره‌گذاری و ارسال مجدد هستند، TCP و جلوگیری از شلوغی شبکه را که معمولاً در UDP این کار پیاده‌سازی می‌کند منعطف تبدیل کرده که برای برنامه‌های امروزی وب بسیار مناسب است.

## اتصالات کارآمد و مقاوم

ساختار QUIC به آن اجازه می‌دهد تا سریع‌تر و بهتر از TCP اتصال برقرار کند. این پروتکل برای بیشتر اتصالات جدید، فقط به یک رفت و برگشت اطلاعات (1-RTT) نیاز دارد و برای اتصالات دوباره برقرار شده، حتی به رفت و برگشت (0-RTT) هم نیاز نیست، که باعث می‌شود ارتباط خیلی سریع‌تر شروع شود. علاوه بر این، قابلیت منحصر به فردی به نام "انتقال اتصال" دارد. این ویژگی به دستگاه‌ها اجازه می‌دهد بدون قطع شدن ارتباط، بین آدرس‌های IP و پورت‌های مختلف جابه‌جا شوند (مثلاً وقتی از Wi-Fi به داده موبایل تغییر می‌دهید. (این کار تجربه کاربری را بهتر کرده و اتصال را در برابر تغییرات شبکه بسیار پایدارتر می‌کند).

در نهایت، انتخاب UDP به عنوان پایه به QUIC کمک می‌کند تا مشکل ("Head-of-Line Blocking") مسدود شدن سرخط (را که در HTTP/2 مبتنی بر TCP وجود دارد، حل کند. در TCP، اگر یک بسته داده از بین برود یا دیر برسد، بقیه اطلاعات در همان اتصال باید منتظر بمانند تا آن بسته دوباره ارسال شود. اما QUIC با استفاده از مدیریت مستقل جریان‌ها روی UDP، تضمین می‌کند که از دست رفتن یا تاخیر یک بسته در یک جریان روی جریان‌های دیگر تاثیر نمی‌گذارد. این باعث می‌شود عملکرد و سرعت پاسخگویی، مخصوصاً در شبکه‌های با تاخیر زیاد یا مشکلات فراوان، به طور قابل توجهی بهتر شود.

# پیاده‌سازی QUIC در برنامه‌های .NET.

## شروع به کار با Kestrel در .NET با QUIC

در 8.NET و 9، توسعه‌دهندگان می‌توانند به راحتی از پروتکل QUIC استفاده کنند. این کار از طریق پشتیبانی داخلی HTTP/3 در Kestrel (Web Server in ASP.NET Core) انجام می‌شود. با این ویژگی، برنامه‌های وب می‌توانند از مزایای QUIC مانند اتصال سریع‌تر، ارسال هم‌زمان داده‌ها بدون مشکل تأخیر (Head-of-Line Blocking) و پایداری اتصال در زمان تغییر شبکه (مثل وای‌فای به موبایل (بهره ببرند).

فعال کردن HTTP/3 در Kestrel خیلی آسان است. کافی است `HttpProtocols.Http3` را به تنظیمات پروتکل اضافه کنید و مطمئن شوید که HTTPS به درستی تنظیم شده باشد. بعد از این کار، Kestrel می‌تواند از اتصالات QUIC استفاده کند. این قابلیت مخصوصاً برای برنامه‌هایی که به ارتباط سریع و حجم بالای داده نیاز دارند، مانند API‌ها، پخش ویدئو و بازی‌های آنلاین، بسیار مهم است.

```
builder.WebHost.ConfigureKestrel(options =>{
    options.ListenAnyIP(5001,
    listenOptions => {
        listenOptions.Protocols =
        HttpProtocols.Http1AndHttp2AndHttp3; // HTTP/1 و HTTP/2 در کنار HTTP/3 فعال کردن
        listenOptions.UseHttps(); // HTTPS برای HTTP/3 نیاز به است
    });
});
```

این تنظیمات به Kestrel می‌گوید که در آدرس IP و پورت 5001 منتظر درخواست‌ها باشد و از HTTP/1، HTTP/2 و HTTP/3 پشتیبانی کند. استفاده از HTTPS برای HTTP/3 ضروری است، چون QUIC خودش رمزنگاری دارد و برای تأیید سرور به گواهی SSL/TLS نیاز دارد.

<https://www.linkedin.com/in/mhramini/>  
<https://github.com/MohamadHoseinRoohiAmini>

## استفاده از HttpClient در سمت کلاینت با QUIC

در سمت کلاینت، .NET این امکان را می‌دهد که با استفاده از HTTP/3 ارتباط برقرار کنید. این تنظیمات به برنامه‌های کلاینت کمک می‌کند تا از بهبودهای QUIC در درخواست‌های خروجی خود استفاده کنند.

```
using var client = new HttpClient();
client.DefaultRequestVersion =
    HttpVersion.Version30; // HTTP/3
client.DefaultRequestPolicy =
    HttpVersionPolicy.RequestVersionOrLower; // در صورت عدم پشتیبانی، به نسخه‌های
                                            // پیش‌تر اشاره کنید
var response = await client.GetAsync("https://example.com"); // قبلی بر می‌گردد
```

تنظیم `DefaultRequestVersion` به `HttpVersion.Version30` روی `HttpClient` می‌گوید که HTTP/3 را ترجیح دهد. اما، استفاده از `DefaultRequestPolicy.RequestVersionOrLower` مطمئن می‌شود که اگر سرور از QUIC پشتیبانی نکند یا مشکلی در اتصال وجود داشته باشد، کلاینت به طور خودکار به HTTP/2 یا HTTP/1.1 برگرد. این روش باعث می‌شود برنامه شما در شبکه‌های مختلف انعطاف‌پذیر و سازگار باشد.

این استفاده در سمت کلاینت برای کارهایی که نیاز به دسترسی سریع و مطمئن به منابع وب دارند، مثل دانلود فایل‌های بزرگ، ارتباط با API‌های ابری و وب‌گردی، بسیار مهم است. با فعال کردن HTTP/3، برنامه‌های .NET می‌توانند تجربه کاربری روان‌تر و پاسخگو‌تر را حتی در شبکه‌های ناپایدار یا با تأخیر زیاد ارائه دهند.

## معرفی System.Net.Quic

جدا از استفاده‌های HTTP/3.NET، یک ابزار سطح پایین‌تر به نام `System.Net.Quic` را نیز ارائه می‌دهد. این ابزار به توسعه‌دهندگان اجازه می‌دهد تا مستقیماً با پروتکل QUIC کار کنند و اتصالات و جریان‌های QUIC را بدون نیاز به مدیریت کنند. این قابلیت برای سناریوهای خاصی مثل ساخت پروتکل‌های سفارشی روی QUIC، ارتباطات بین دو برنامه (peer-to-peer)، یا توسعه سرویس‌هایی که نیاز به کنترل دقیق بر ویژگی‌های QUIC مانند مدیریت جریان و Datagram دارند، بسیار مفید است. `System.Net.Quic` در .NET 9. کامل‌تر شده و گزینه‌های پیشرفت‌تری برای سفارشی‌سازی و بهینه‌سازی ارائه می‌دهد.

# در .NET 9 System.Net.Quic



(۱۵)

## QuicListener

این بخش سمت سرور به درخواست‌های جدید QUIC گوش می‌دهد و پورت UDP را مدیریت می‌کند. وظیفه آن پذیرش اتصالات جدید و آماده‌سازی آن‌ها برای پردازش است. این کلاس پایه محکمی برای ساخت سرورهای بسیار کارآمد و سریع با استفاده از QUIC فراهم می‌کند. توسعه‌دهندگان می‌توانند از آن برای ساخت پروتکل‌های خاص خود یا سرویس‌هایی که نیاز به استفاده مستقیم از قابلیت‌های پیشرفته QUIC دارند، استفاده کنند.

## QuicConnection

این کلاس هر اتصال QUIC را مدیریت می‌کند، از لحظه شروع و امن کردن ارتباط تا نگهداری و قطع آن. قابلیت‌هایی مانند "مهاجرت اتصال" را QuicConnection فراهم می‌کند. این یعنی کاربران می‌توانند آدرس IP اخود را تغییر دهند (مثلاً از وای‌فای به داده موبایل) (بدون اینکه اتصال قطع شود، که برای برنامه‌های موبایل و خدماتی که باید همیشه فعال باشند، بسیار مهم است).

## QuicStream

این کلاس جریان‌های جداگانه داده را درون یک اتصال QUIC مدیریت می‌کند. این امکان را فراهم می‌کند که داده‌ها به صورت دوطرفه یا یک‌طرفه منتقل شوند و از ویژگی مهم QUIC یعنی "مالتی‌پلکسینگ" بهره می‌برد. این یعنی چندین جریان داده می‌توانند همزمان و بدون اینکه یکی جلوی دیگری را بگیرد (مشکل انسداد سر خط (در یک اتصال ارسال شوند، که باعث کاهش تاخیر می‌شود).

این کلاس‌ها به توسعه‌دهندگان کنترل کامل و دقیق بر ارتباطات QUIC می‌دهند، که فراتر از HTTP/3 است. این امکان به برنامه‌نویسان اجازه می‌دهد تا از تمام ویژگی‌های پیشرفته QUIC برای ساخت برنامه‌های شبکه‌ای بسیار بهینه و سریع استفاده کنند. با استفاده مستقیم از این ابزارها، می‌توان پروتکل‌های خاصی ساخت، عملکرد را بهینه کرد و از مزایای ذاتی QUIC مانند رمزنگاری داخلی و قابلیت اطمینان بالاتر بهره برد.

برای مثال، می‌توان از این کلاس‌ها در ساخت سرورهای بازی با تاخیر بسیار کم، سیستم‌های پخش ویدئو با کیفیت بالا یا هر سرویس دیگری که نیاز به انتقال داده‌های کارآمد و قابل اعتماد بر

بستر UDP دارد، استفاده کرد. این قابلیت‌ها System.Net.Quic را به ابزاری قدرتمند در اختیار توسعه‌دهندگان.NET تبدیل می‌کنند تا چالش‌های شبکه‌ای امروزی را حل کنند.

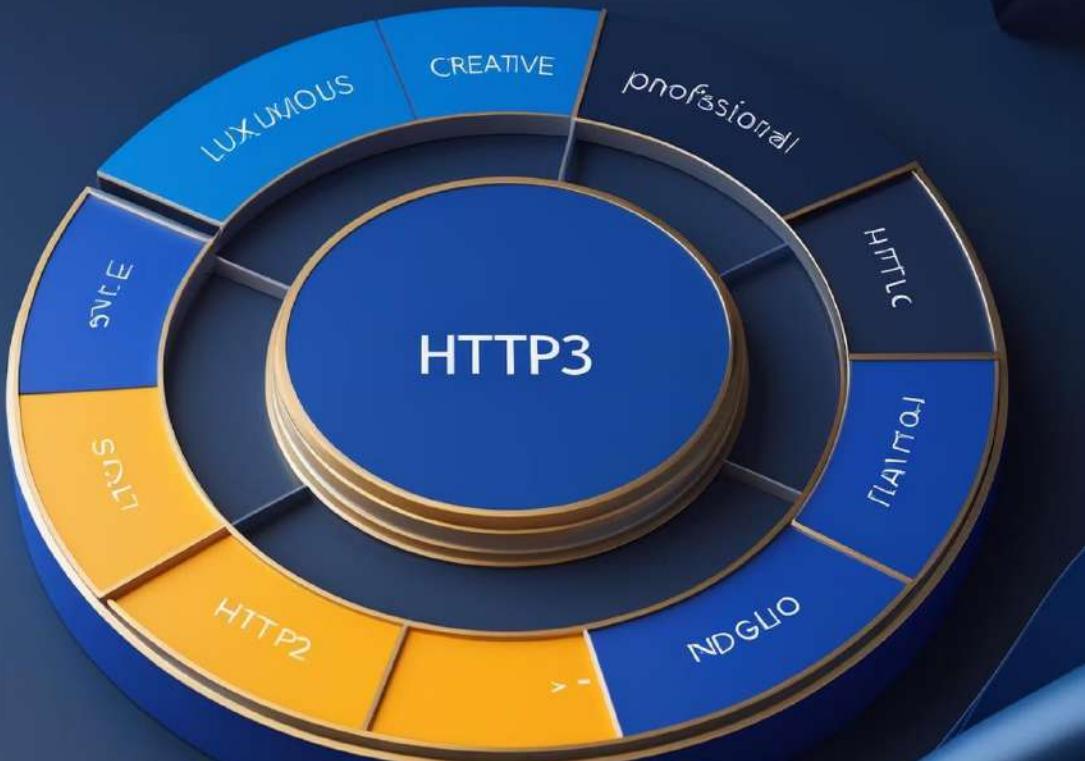
<https://www.linkedin.com/in/mohamad-hosein-roohi-amini/>

<https://github.com/MohamadHoseinRoohiAmini>

# مزایا و چالش‌های عملکرد

## چالش‌ها

## مزایا



- شروع سریع اتصال 0-RTT و 1-RTT: QUIC سریع‌تر به سرور وصل می‌شود. دفعه اول کمی طول می‌کشد، اما دفعات بعدی تقریباً بلا فاصله (0-RTT) وصل می‌شوند.
- رفع مشکل توقف جریان داده (Head-of-Line Blocking): QUIC، اگر یک بسته اطلاعاتی گم شود، بقیه جریان‌های داده متوقف نمی‌شوند. این باعث می‌شود سرعت انتقال اطلاعات بالاتر برود.
- تغییر شبکه بدون قطع شدن اتصال Connection Migration: اگر شبکه شما عوض شود (مثلاً از وای‌فای به داده موبایل برود)، اتصال QUIC شما قطع نمی‌شود و کارتان ادامه پیدا می‌کند. این برای گوشی‌های موبایل خیلی خوب است.
- امنیت بیشتر با رمزگذاری داخلی: همه اطلاعات در QUIC صورت خودکار رمزگذاری می‌شوند (با استفاده از TLS 1.3). باعث می‌شود که امنیت بیشتری داشته باشید و دستکاری اطلاعات سخت‌تر شود.
- جریان‌های مستقل داده و استفاده بهتر از منابع QUIC: اجازه می‌دهد چندین جریان داده را همزمان و جداگانه ارسال کنید. این کار باعث می‌شود از اینترنت بهتر استفاده شود و برنامه‌ها سریع‌تر کار کنند.
- عملکرد عالی در شبکه‌های ضعیف QUIC: حتی در اینترنت‌های کند یا جاهایی که بسته‌های اطلاعاتی گم می‌شوند، بهتر از پروتکل‌های قدیمی‌تر عمل می‌کند.
- کنترل بهتر ترافیک QUIC: می‌تواند ترافیک اینترنت را به روش‌های هوشمندانه‌تری مدیریت کند تا پهنه‌ای باند بهتر و کارآمدتر استفاده شود.

- پشتیبانی محدود و جدید بودن فناوری: هنوز همه جا از QUIC پشتیبانی نمی‌شود، مثلًا در بعضی فایروال‌ها یا سیستم‌عامل‌ها. این باعث می‌شود استفاده از آن در همه جا سخت باشد.
- محدودیت در مرورگرها و ابزارهای توسعه: بعضی مرورگرها ممکن است در استفاده از HTTP/3 در محیط‌های توسعه (مثل localhost) محدودیت داشته باشند. همچنین برای دیدن اطلاعات رمزگذاری شده QUIC به ابزارهای خاصی نیاز است.
- مصرف بیشتر CPU: چون QUIC بیشتر در نرم‌افزار اجرا می‌شود و رمزگذاری همیشه روشن است، ممکن است نسبت به TCP که بیشتر در هسته سیستم‌عامل اجرا می‌شود (انرژی بیشتری از پردازنده مصرف کند).
- مشکل با فایروال‌ها و NAT: به دلیل استفاده از UDP، ممکن است بعضی فایروال‌ها یا دستگاه‌های NAT با QUIC مشکل داشته باشند و نیاز به تنظیمات خاصی باشد.
- نیاز به پشتیبانی Fallback: برنامه‌هایی که از QUIC استفاده می‌کنند باید بتوانند در صورت عدم پشتیبانی از QUIC، به پروتکل‌های قدیمی‌تر مثل HTTP/2 برگردند تا همیشه اتصال برقرار باشد.
- نیاز به یادگیری برای برنامه‌نویسان: برنامه‌نویسانی که فقط با TCP کار کرده‌اند، ممکن است برای درک ویژگی‌های جدید QUIC مثل "شناسه اتصال" نیاز به زمان بیشتری برای یادگیری داشته باشند.
- دشواری در رصد و رفع اشکال: چون اطلاعات QUIC رمزگذاری شده است، دیدن و بررسی ترافیک آن برای ابزارهای سنتی رصد شبکه سخت‌تر است.

# روندهای آینده در شبکه‌سازی

در ادامه به برخی از روندهای مهم آینده در این پروتکل‌ها با ویژگی‌های جدید خود، برای پشتیبانی از برنامه‌ها و سرویس‌های وب آینده که نیاز به سرعت، امنیت و انعطاف‌پذیری بالایی دارند، بسیار مهم می‌شوند. در حال تغییر دادن اینترنت هستند HTTP/3 و QUIC.

## برنامه‌های بلادرنگ (Real-time)

تأثیر کم QUIC به برنامه‌های بلادرنگ مثل تماس تصویری با کیفیت بالا، بازی‌های آنلاین، واقعیت افزوده (AR)، واقعیت مجازی (VR) و ابزارهای همکاری گروهی کمک می‌کند تا سریع‌تر عمل کند. قابلیت‌های QUIC برای جلوگیری از تأخیر در انتقال داده و استفاده از جریان‌های مستقل، تضمین می‌کند که اطلاعات با کمترین تأخیر منتقل شوند، که برای ارتباطات فوری و روان در این نوع برنامه‌ها ضروری است.

## اتصال اینترنت اشیا (IoT)

کارایی و انعطاف‌پذیری QUIC آن را برای دستگاه‌های اینترنت اشیا (IoT) که اغلب در شرایط شبکه دشوار و با منابع محدود کار می‌کنند، مناسب می‌سازد. نیاز کم به پردازش و توانایی آن برای کارکرد مؤثر در شبکه‌هایی که بسته‌ها گم می‌شوند، QUIC و 5G همکاری. توانایی پشتیبانی از تعداد زیادی اتصال همزمان بهره می‌برد به کاربران امکان می‌دهد تا تجربه‌های بین‌ظیری در زمینه‌هایی مانند واقعیت مجازی متصل، خودروهای خودران و شهرهای هوشمند داشته باشند، جایی که ارتباطات سریع و قابل اعتماد ضروری است.

## امنیت پیشرفته

رمزگذاری و احراز هویت داخلی TLS با 1.3 استانداردهای جدیدی را برای ارتباطات امن در اینترنت تعیین می‌کند. این روش رمزگذاری پیش‌فرض، حملات شنود و دستکاری را کاهش می‌دهد و تضمین می‌کند که داده‌ها از زمان ارسال تا رسیدن به مقصد امن بمانند. این کار نه تنها حریم خصوصی کاربران را افزایش می‌دهد، بلکه به ایجاد اینترنتی قابل اعتمادتر برای همه کمک می‌کند و حتی در برابر تهدیدات امنیتی جدید، از جمله آنها که ممکن است با محاسبات کوانتومی ظاهر شوند، مقاوم‌تر است.



## تجربه بهتر موبایل

قابلیت تغییر اتصال QUIC آن را برای برنامه‌های موبایل که نیاز به اتصال بدون قطعی دارند، عالی می‌کند؛ به ویژه زمانی که کاربران بین شبکه‌ها جابجا می‌شوند. این یعنی تجربه کاربری یکپارچه‌تر، قطعی‌های کمتر و عملکرد بهتر در شبکه‌های ضعیف، که مستقیماً روح رضایت کاربر را تاثیر می‌گذارد. همچنین، کاهش نیاز به شروع مجدد اتصال و کارایی بالاتر به صرفه‌جویی در مصرف باتری دستگاه‌های موبایل کمک می‌کند.

## ادغام با شبکه‌های 5G و آینده

و نسل‌های آینده طراحی شده 5G برای استفاده کامل از پتانسیل شبکه‌های QUIC مانند تأخیر بسیار کم، پهنای باند بالا و 5G این پروتکل از ویژگی‌های مهم است. QUIC به کاربران امکان می‌دهد تا تجربه‌های بین‌ظیری در زمینه‌هایی مانند واقعیت مجازی متصل، خودروهای خودران و شهرهای هوشمند داشته باشند، جایی که ارتباطات سریع و قابل اعتماد ضروری است.

## معماری‌های بدون سرور و Edge Computing

با زمان‌های اتصال سریع‌تر و قابلیت‌های همزمان‌سازی کارآمد خود، به خوبی با Edge Computing معماری‌های بدون سرور و سریع‌تر و مؤثرتر بین توایع بدون سرور و گره‌های لبه‌ای را فراهم می‌کند، که منجر به این همکاری به کاهش تأخیر و پاسخ‌گویی بهتر در برنامه‌های توزیع‌شده می‌شود توسعه راه حل‌های ابری نسل آینده کمک می‌کند که نیاز به منابع محاسباتی نزدیک‌تر به کاربر نهایی دارند.

# جمع‌بندی مفاهیم شبکه

## استفاده از QUIC در داتنت

جامعه برنامه‌نویسان داتنت از پشتیبانی خوبی برای تکنولوژی‌های شبکه مدرن برخوردار است. با اضافه شدن پشتیبانی مستقیم از HTTP/3 و QUIC در نسخه‌های جدیدتر (NET.8 و NET.9)، برنامه‌نویسان می‌توانند به راحتی از سرعت و امنیت این پروتکل‌ها در برنامه‌های خود استفاده کنند. کلاس‌های موجود در بخش System.Net.Quic کنترل دقیقی بر ارتباطات QUIC می‌دهند و به برنامه‌نویسان اجازه می‌دهند تا سناریوهای پیچیده‌تر شبکه را مدیریت کنند. این پشتیبانی یکپارچه، ساخت برنامه‌های سریع، انعطاف‌پذیر و امن را در داتنت آسان‌تر می‌کند و به آن‌ها کمک می‌کند تا با پیشرفت‌های آینده شبکه همگام شوند.

## چرا QUIC مهم است؟

یک پروتکل جدید برای انتقال اطلاعات است که QUIC در شبکه‌های امروزی TCP مخصوصاً برای حل مشکلات یکی از مهمترین ویژگی‌های آن، چندین جریان داده همزمان است که به Stream Multiplexing (آغازه می‌دهد روی یک ارتباط و به صورت QUIC چندین داده) این کار از مشکل کندی ناشی از "Head-of-Line Blocking" (جداگانه منتقل شوند) می‌شود (مشخص شد. حتی HTTP/2 که سعی کرد این مشکلات را حل کند، باز هم روی TCP کار می‌کرد و محدودیت‌های آن را داشت). این پیشرفت کم کم به QUIC و HTTP/3 رسید. این پروتکل‌ها با روشی جدید بر پایه UDP، راه را برای ارتباطات سریع‌تر، امن‌تر و با انعطاف‌پذیری بیشتر باز کردند.

است که به دستگاه‌ها Connection Migration (اتصال اجازه می‌دهد بدون قطع شدن ارتباط، آدرس اینترنتی یا پورت این ویژگی برای گوشی‌های همراه و خود را عوض کنند همچنین، کسانی که در حال حرکت هستند، خیلی مهم است با استفاده از Integrated Encryption (رمزنگاری داخلی)، امنیت را از همان ابتدا وارد پروتکل کرده و زمان TLS 1.3 را بسیار کوتاه می‌کند و سرعت Handshake (آغاز ارتباط) را این ترکیب خاص از ویژگی‌ها، اتصال را بالا می‌برد. انتخابی عالی برای برنامه‌های جدید کرده است

## پروتکل‌های شبکه چگونه بهتر شدند؟

در ابتدا، برای انتقال مطمئن و سریع اطلاعات، پروتکل‌های مثل TCP/IP ساخته شدند که پایه اینترنت امروزی هستند. اما وقتی اینترنت بزرگتر شد و برنامه‌های پیچیده‌تری آمدند، مشکلات پروتکل‌های قدیمی‌تر مثل HTTP/1.x و TCP/IP (مثلًا مشکل "Head-of-Line Blocking" که باعث کندی مشخص شد. حتی HTTP/2 که سعی کرد این مشکلات را حل کند، باز هم روی TCP کار می‌کرد و محدودیت‌های آن را داشت. این پیشرفت کم کم به QUIC و HTTP/3 رسید. این پروتکل‌ها با روشی جدید بر پایه UDP، راه را برای ارتباطات سریع‌تر، امن‌تر و با انعطاف‌پذیری بیشتر باز کردند).

با شناخت کامل پروتکل‌های شبکه جدید مانند QUIC و HTTP/3، برنامه‌نویسان نه تنها می‌توانند برنامه‌هایی بسازند که نیازهای ارتباطی امروز را برطرف می‌کنند، بلکه می‌توانند تجربه کاربری بهتری را در شبکه‌های پیچیده آینده ارائه دهند. این دانش برای نوآوری در زمینه‌هایی مثل برنامه‌های مخصوص موبایل، ارتباطات زنده (Real-time)، دستگاه‌های اینترنت اشیا و راه حل‌های امنیتی پیشرفت‌هایی که به سرعت در حال رشد هستند، ضروری است.

<https://www.linkedin.com/in/mhramini/>

<https://github.com/MohamadHoseinRoohiAmini>