

ارائه: چت‌روم ساده با WebSocket

عنوان پروژه

پروژه مهندسی اینترنت

چت‌روم ساده با WebSocket

یک نمونه آموزشی برای ارسال و دریافت پیام هم‌زمان بین چند کاربر.

متصل

پاک کردن

اتاق: مهندسی اینترنت

شما با نام علی گفتگو می‌کنید.

سیستم
۱۴:۳۸
علی وارد اتاق مهندسی اینترنت شد.

ارسال

پیام خود را بنویسید...

ورود به اتاق

نام شما

علی

نام اتاق

مهندسی اینترنت

قطع اتصال

اتصال

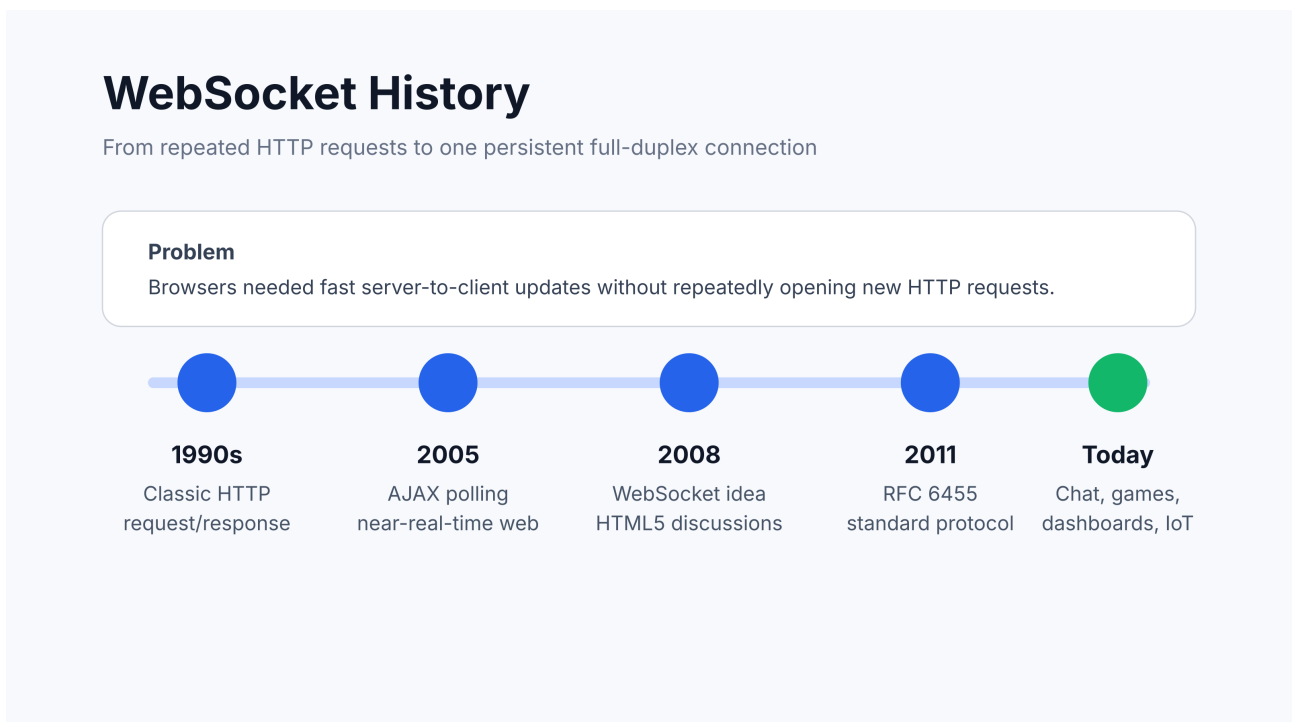
کاربران آنلاین

علی

چت‌روم ساده با WebSocket

این پروژه برای درس مهندسی اینترنت آماده شده است و هدف آن نمایش ارتباط بلادرنگ بین مرورگر و سرور با استفاده از FastAPI و WebSocket است.

چرا WebSocket به وجود آمد؟



وب در ابتدا بیشتر بر پایه مدل درخواست و پاسخ HTTP کار می‌کرد. مرورگر یک درخواست می‌فرستاد، سرور پاسخ می‌داد و ارتباط همان‌جا تمام می‌شد.

با پیچیده‌تر شدن برنامه‌های وب، نیاز به دریافت سریع اطلاعات جدید از سمت سرور بیشتر شد؛ مثلاً پیام جدید در چت، اعلان‌ها یا وضعیت زنده یک سیستم.

قبل از WebSocket معمولا از روش‌هایی مثل polling و polling long استفاده می‌شد. این روش‌ها قابل استفاده بودند، اما باعث تکرار زیاد درخواست‌های HTTP و مصرف منابع بیشتر می‌شدند.

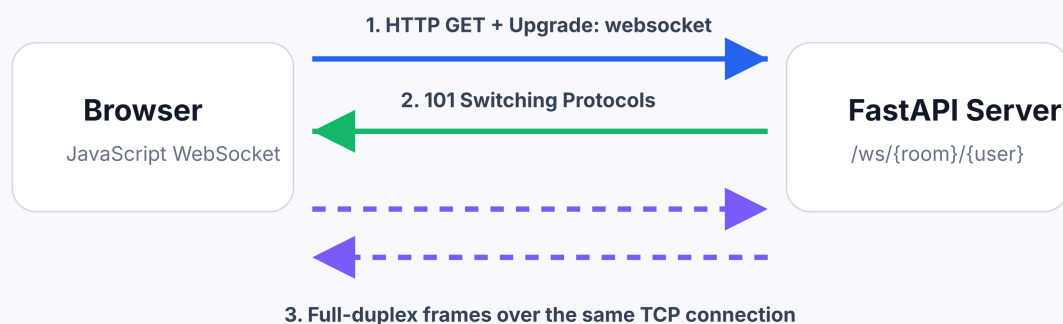
تاریخچه کوتاه

- **HTTP کلاسیک:** برای دریافت صفحه و ارسال فرم مناسب بود، اما برای ارسال لحظه‌ای داده از سرور به مرورگر طراحی نشده بود.
- **AJAX و polling:** امکان درخواست‌های پس‌زمینه را فراهم کرد، ولی مرورگر همچنان باید مرتب از سرور سوال می‌پرسید.
- **Long polling:** تعداد درخواست‌های بی‌فایده را کمتر کرد، اما همچنان هر پیام جدید درگیر چرخه HTTP بود.
- **استاندارد WebSocket:** در RFC ۶۴۵۵ به عنوان یک پروتکل استاندارد برای ارتباط دوطرفه و پایدار معرفی شد.
- **کاربردهای امروز:** چت، بازی آنلاین، داشبورد زنده، ویرایش هم‌زمان، اعلان‌ها و کنترل تجهیزات IoT.

WebSocket چگونه از HTTP شروع می‌شود؟

HTTP Upgrade to WebSocket

The connection starts as HTTP, then switches protocols after the handshake.



WebSocket از ابتدا یک ارتباط جدا از HTTP نیست. ارتباط با یک درخواست HTTP معمولی شروع می‌شود که از سرور می‌خواهد پروتکل را تغییر دهد. مرورگر یک درخواست GET با سرآیندهایی شبیه این می‌فرستد:

```
Connection: Upgrade
Upgrade: websocket
Sec-WebSocket-Key: ...
```

اگر سرور درخواست را قبول کند، پاسخ زیر را برمی‌گرداند:

```
101 Switching Protocols
```

بعد از این پاسخ، همان اتصال TCP دیگر مثل HTTP عادی استفاده نمی‌شود و داده‌ها در قالب فریم‌های WebSocket جابه‌جا می‌شوند.

بعد از ارتقا چه اتفاقی می‌افتد؟

بعد از تبدیل ارتباط:

- مرورگر هر زمان بخواهد می‌تواند برای سرور پیام بفرستد.
- سرور هم هر زمان بخواهد می‌تواند برای مرورگر پیام بفرستد.
- ارتباط باز می‌ماند و برای هر پیام، درخواست HTTP جدید ساخته نمی‌شود.

• داده‌ها به صورت فریم WebSocket ارسال می‌شوند.

به همین دلیل به WebSocket ارتباط **دوطرفه کامل** گفته می‌شود.

جایگاه WebSocket در مدل OSI

WebSocket in the OSI Model

WebSocket is used by applications, while TCP/IP carries bytes through the network.

Layer 7 - Application	HTTP upgrade, then WebSocket frames
Layer 6 - Presentation	Text, JSON, UTF-8
Layer 5 - Session	Long-lived logical conversation
Layer 4 - Transport	TCP connection, reliable stream
Layer 3 - Network	IP routing between hosts
Layers 2-1 - Link / Physical	Ethernet, Wi-Fi, signals

وب‌سوکت معمولاً در لایه کاربرد بررسی می‌شود، چون برنامه تصمیم می‌گیرد پیام‌ها چه معنایی دارند.

در یک ارتباط معمول بین مرورگر و سرور:

- برنامه از WebSocket برای ارسال و دریافت پیام استفاده می‌کند.
- شروع ارتباط با مفاهیم HTTP انجام می‌شود.
- انتقال قابل اعتماد داده‌ها بر عهده TCP است.
- IP بسته‌ها را بین میزبان‌ها مسیریابی می‌کند.
- Ethernet یا Wi-Fi داده را روی شبکه واقعی حمل می‌کند.

نگاه شبکه‌ای ساده

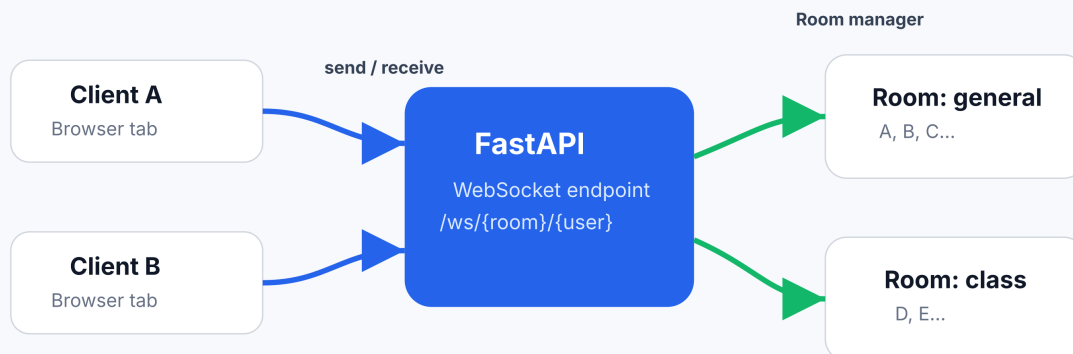
مسیر داده در این پروژه را می‌توان این‌طور ساده کرد:

JSON قالب در چت پیام
WebSocket فریم
TCP قطعه
IP بسته
شبکه فریم
فیزیکی سیگنال

کد پروژه بیشتر در بالاترین بخش این زنجیره کار می‌کند. پیام JSON ساخته می‌شود، از طریق WebSocket فرستاده می‌شود و جزئیات انتقال در لایه‌های پایین‌تر توسط سیستم‌عامل و شبکه انجام می‌شود.

Project Architecture

Each browser keeps one WebSocket connection. The server broadcasts only inside the selected room.



این پروژه سه بخش اصلی دارد:

- **رابط مرورگر:** فایل‌های CSS، HTML و JavaScript.
- **مسیر WebSocket:** مسیر FastAPI با الگوی `./ws/{room_id}/{client_id}`.
- **مدیر اتصال‌ها:** نگهداری های socket فعال به تفکیک اتاق.

وقتی یک کاربر پیام می‌فرستد، سرور پیام را از socket همان کاربر می‌گیرد و آن را برای همه های socket همان اتاق پخش می‌کند.

جریان پیام در پروژه

۱. کاربر نام و اتاق را وارد می‌کند.
۲. جاوااسکریپت یک شیء WebSocket می‌سازد.
۳. مرورگر درخواست Upgrade HTTP را ارسال می‌کند.
۴. برنامه‌ی FastAPI اتصال WebSocket را قبول می‌کند.
۵. مدیر اتصال، کاربر را در لیست اتاق ذخیره می‌کند.
۶. پیام به صورت JSON از مرورگر به سرور فرستاده می‌شود.
۷. سرور پیام را برای کاربران همان اتاق پخش می‌کند.
۸. مرورگرها پیام جدید را بدون تازه‌سازی صفحه نمایش می‌دهند.

Pros and Cons

WebSocket is powerful for real-time systems, with some server and network tradeoffs.

Pros

- Low latency updates
- Full-duplex communication
- Less HTTP overhead
- Great for chat and dashboards

Cons

- Long-lived connections use memory
- Scaling needs shared state
- More connection failure cases
- Not ideal for simple CRUD pages

وب‌سوکت برای سیستم‌های بلادرنگ بسیار کاربردی است، اما برای همه صفحه‌های وب انتخاب لازم یا مناسب نیست.

• مزایا

- تاخیر کم، چون ارتباط باز می‌ماند.
- امکان ارسال فوری داده از سمت سرور به مرورگر.
- مناسب برای چت، اعلان، ابزارهای همکاری، بازی و داشبورد زنده.
- کاهش سربار سرآیندهای تکراری HTTP بعد از دست‌دهی اولیه.
- مدل برنامه‌نویسی ساده برای ارتباط پیوسته و دوطرفه.

• معایب

- اتصال‌های طولانی‌مدت از منابع سرور استفاده می‌کنند.
- مقیاس‌پذیری روی چند سرور به طراحی بیشتر نیاز دارد.
- قطع اتصال، اتصال دوباره و خطاهای شبکه باید مدیریت شوند.
- بعضی پراکسی‌ها یا دیواره‌های آتش ممکن است تنظیم درست برای WebSocket بخواهند.
- برای صفحه‌های ساده CRUD معمولاً لازم نیست.

برنامه اجرای دمو

۱. سرور را با دستور زیر اجرا کنید:

```
uvicorn app.main:app --reload
```

۲. برنامه را در دو تب مرورگر باز کنید.
۳. در هر تب یک نام متفاوت وارد کنید.
۴. هر دو کاربر وارد یک اتاق شوند.
۵. از یک تب پیام ارسال کنید.
۶. دریافت فوری پیام در تب دوم را نشان دهید.
۷. یکی از کاربران را قطع کنید و تغییر فهرست کاربران آنلاین را نمایش دهید.