



Ciência da **Computação**

Programação Orientada a Objetos Avançado
Prof. Luciano Rodrigo Ferretto

Autenticação e Autorização em APIs Rest



Authorization

What you can do



Authentication

Who you are

Autenticação

- A autenticação e a autorização são dois aspectos cruciais da segurança em qualquer aplicativo, especialmente em APIs REST (Representational State Transfer). Elas garantem que apenas usuários autorizados tenham acesso aos recursos e funcionalidades da API, protegendo os dados sensíveis e mantendo a integridade do sistema.
- A autenticação é o processo de verificar a identidade do usuário que está tentando acessar a API. Em APIs REST, a autenticação geralmente é baseada em tokens de acesso (access tokens) ou em credenciais fornecidas pelo cliente.
 - Existem várias formas de autenticação, como autenticação básica, autenticação baseada em token, OAuth, entre outras.

Autorização

- Após a autenticação, entra em jogo a autorização. A autorização determina quais recursos e ações um usuário autenticado pode acessar na API. Ela é baseada nos privilégios e permissões associados ao usuário autenticado. As permissões podem ser definidas com base em papéis (roles) ou em níveis de acesso específicos para cada recurso.
- Para implementar a autorização em APIs REST, é comum usar listas de controle de acesso (ACLs), tokens de acesso com escopo limitado ou estruturas mais avançadas, como o OAuth 2.0. O OAuth 2.0 é um protocolo de autorização amplamente utilizado, que permite que um aplicativo cliente acesse recursos em nome do usuário, sem compartilhar suas credenciais reais. Ele fornece um fluxo de autorização seguro e flexível para API REST.

Conclusão

- Em resumo, a autenticação e a autorização em APIs REST são essenciais para garantir a segurança dos dados e controlar o acesso aos recursos. A autenticação verifica a identidade do usuário, enquanto a autorização define as permissões e privilégios para esse usuário. A combinação correta desses mecanismos proporciona uma base sólida para proteger sua API REST contra acesso não autorizado e outros ataques.

Mas antes...

Precisamos entender a diferença
entre comunicação **Statefull** e
Stateless

Imagine você ligando para seu banco para solicitar um empréstimo.



Conexão Statefull

- Este é um bom exemplo de conexão **stateful**, ou seja, uma comunicação **com estado**.
- Após a cliente confirmar os seus dados, o atendimento continua e, enquanto a ligação estiver ativa, o atendente saberá que está falando com **Ana Júlia**.
Ele não precisa perguntar novamente quem é ela a cada nova pergunta ou informação trocada.
- O contexto foi mantido ao longo da conversa.



Agora imagine que ela
vai solicitar o
empréstimo por e-mail



Conexão Stateless



- Agora, imagine que ao invés de ligar para o banco, Ana Júlia resolve se comunicar por e-mail.
- A cada novo e-mail que ela envia, o banco **não sabe automaticamente que é ela** — é necessário que ela **se identifique novamente**, colocando seus dados ou um número de protocolo, por exemplo.
- Este é o funcionamento de essência do protocolo **HTTP**: Cada requisição feita ao servidor é **independente**. O servidor **não guarda memória de requisições anteriores**, então **o cliente precisa enviar tudo novamente a cada requisição** — inclusive quem ele é!

HTTP – Processo de comunicação em rotas protegidas – Conexões Stateless

- Imagine que o HTTP é como enviar e-mail ou cartas pelo correios.
- Quando você pede algo ao servidor, é como enviar uma carta solicitando informações (REQUEST). O servidor responde com outra carta contendo as informações que você pediu (RESPONSE).
- Cada “carta” é preciso ter todas as informações para identificar os envolvidos na comunicação, exemplo o remetente e o destinatário.

DESTINATÁRIO

MARIA FRANCISCA MENDES ROCHA AZEVEDO
Rua Barão de Campinas, nº 400 - apto 39
Campos Elíseos - São Paulo - SP
Cep: 01201-000.



REMETENTE

JOÃO FRANCISCO MENDES ROCHA AZEVEDO
Rua Maria José, nº 470 - apto 19
Bela Vista - São Paulo - SP
Cep: 01324-010.

HTTP – Processo de comunicação em rotas protegidas – Conexão Stateless

- HTTP **não** mantém uma “linha telefônica constante” entre você e o servidor.
- Ou seja, não é como uma ligação telefônica que permanece aberta o tempo todo. Cada vez que você pede algo, é como fazer uma nova ligação.
- Isso significa que o servidor não se lembra de você entre uma "ligação" e outra.


HTTP – Processo de comunicação em rotas protegidas – Conexão Stateless

- Então, se você está acessando uma rota protegida, o servidor precisa de uma maneira de saber que é você novamente.
- É como se, a cada nova carta que você envia pedindo algo protegido, você tivesse que incluir uma senha ou algum tipo de identificação para que o servidor saiba que é você mesmo.
- **Moral da história:** Por ser **stateless naturalmente**, o HTTP **precisa de mecanismos extras de identificação**, como autenticação com tokens, cookies, sessões ou cabeçalhos especiais — senão, o servidor nunca saberá **quem** está acessando suas rotas protegidas.

Autenticação básica

- A **Autenticação Básica (Basic Auth)** é uma forma simples de ilustrar como funciona o processo de identificação do usuário em uma requisição HTTP — ideal para começar antes de introduzir JWT ou OAuth.
- No entanto, esse método **não é considerado seguro para uso em APIs REST**, pois as credenciais são enviadas em texto puro e podem ser interceptadas por um atacante.

Autenticação básica

-  **Mas Base64 não é criptografia!**
- Base64 ≠ Segurança
- A codificação serve apenas para transmitir dados de forma “segura” pela rede, mas não protege contra ataques.
- Por isso, Basic Auth só deve ser usado com HTTPS (e em raríssimos casos), caso contrário, o usuário e senha podem ser facilmente interceptados.

Autenticação básica

- A autenticação básica exige que o cliente envie um **usuário** e **senha** codificados em **Base64**, no cabeçalho *Authorization* da requisição HTTP.

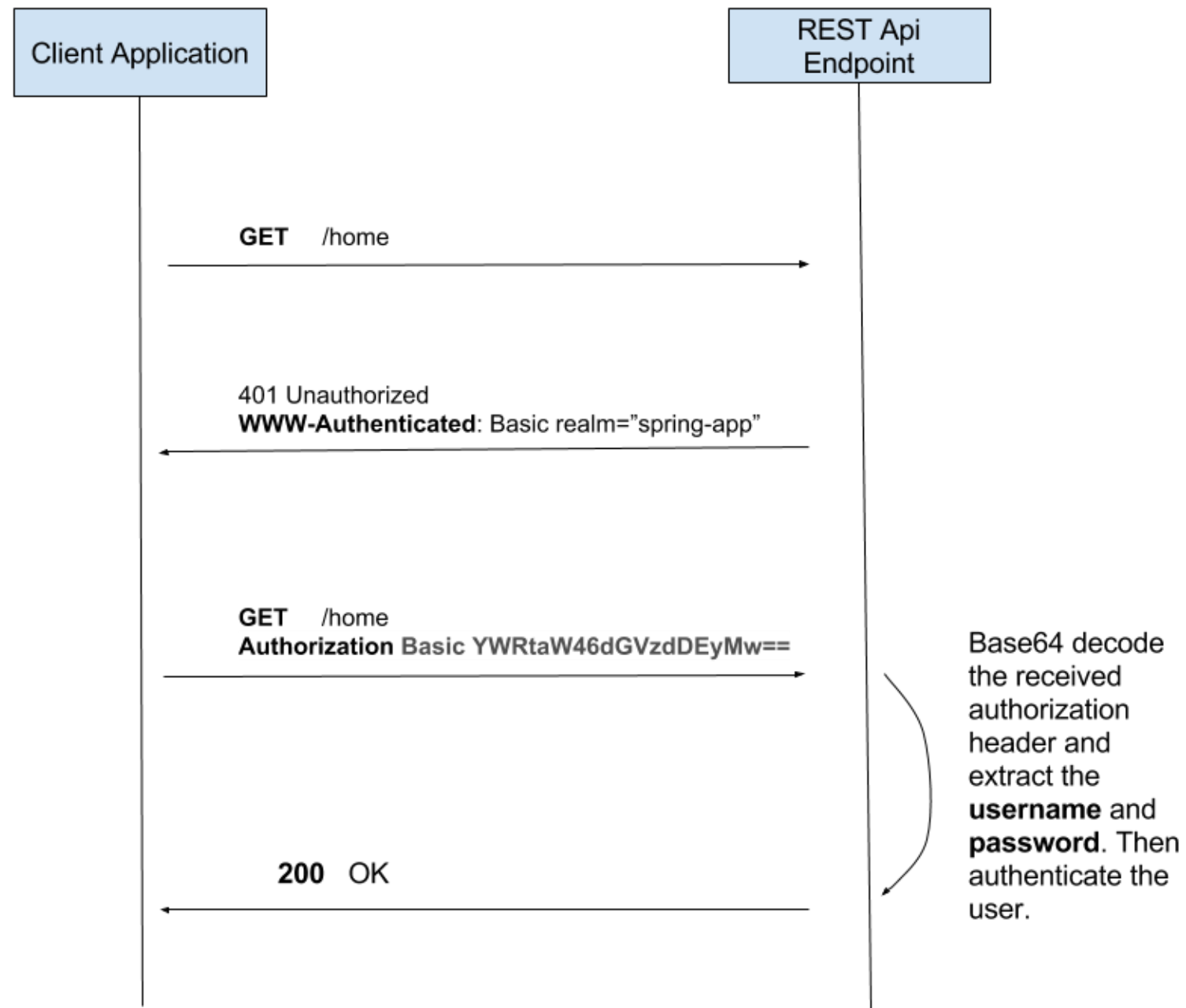
- Exemplo:

```
GET /api/dados-privados HTTP/1.1
```

```
Host: minhaapi.com
```

```
Authorization: Basic YW5hanVsaWE6c2VuaGEzMjM=
```

- O valor após Basic é: *username:password* codificado em Base64.
- No exemplo acima, o valor decodificado é: *anajulia:senha123*





Autenticação via sessão

- A autenticação via sessão é muito comum em aplicações web tradicionais com frontend baseado em HTML (como JSP, Thymeleaf, etc).
- Nesse modelo, o servidor mantém o estado da sessão do usuário na memória ou em um banco de sessão, e o cliente mantém apenas um identificador de sessão (session ID), geralmente em um cookie.

Autenticação via sessão

ATENÇÃO!!!

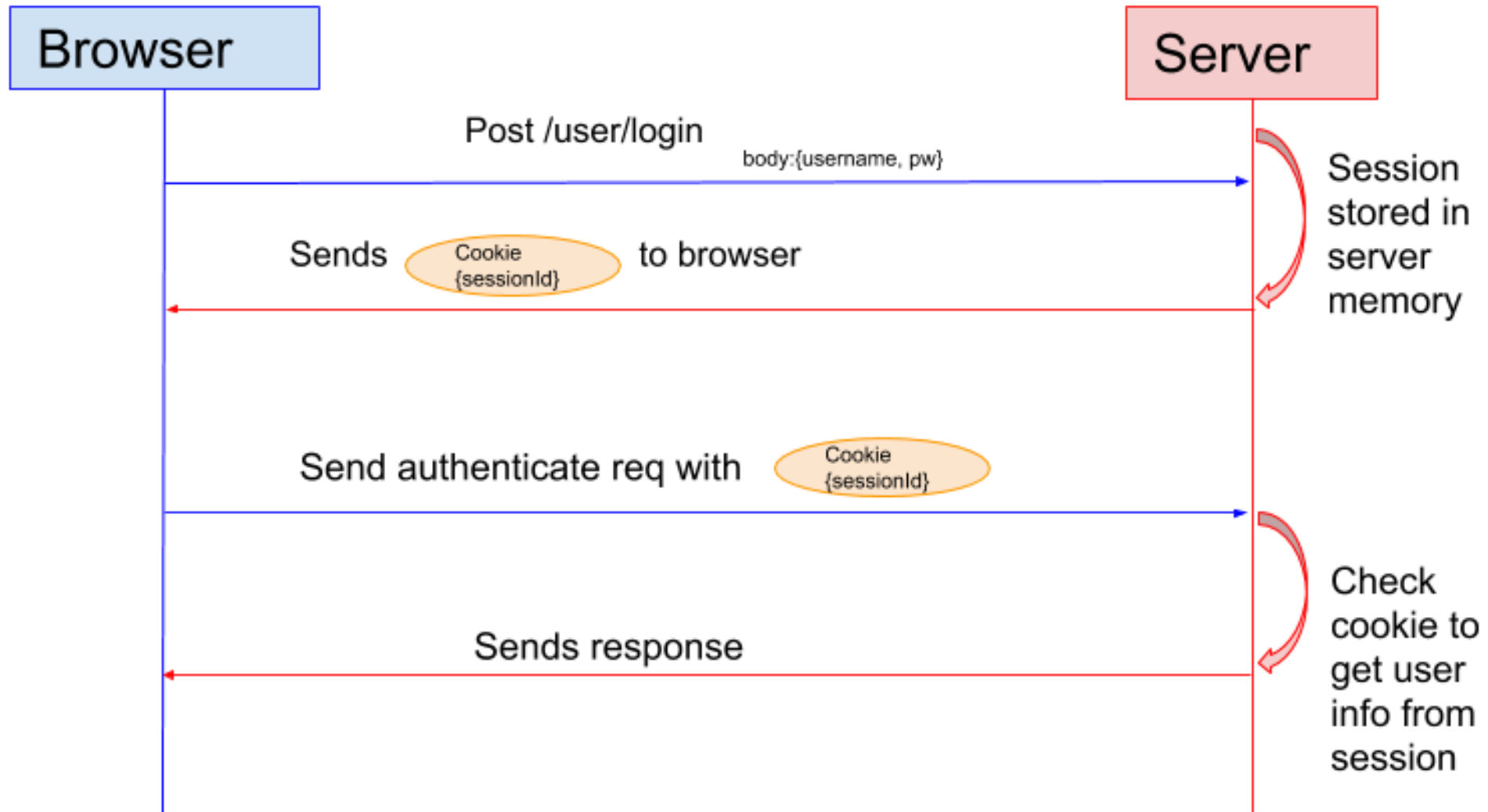
-  O HTTP é stateless por natureza, ou seja, cada requisição é independente.
-  Mas aplicações podem **simular estado** com mecanismos como **cookies, sessões ou tokens**, fazendo com que **o servidor “lembre” do usuário** entre as requisições — mesmo que o protocolo em si não tenha essa capacidade.
- A “statefull” é implementado na camada de aplicação, não no protocolo.

Autenticação via sessão

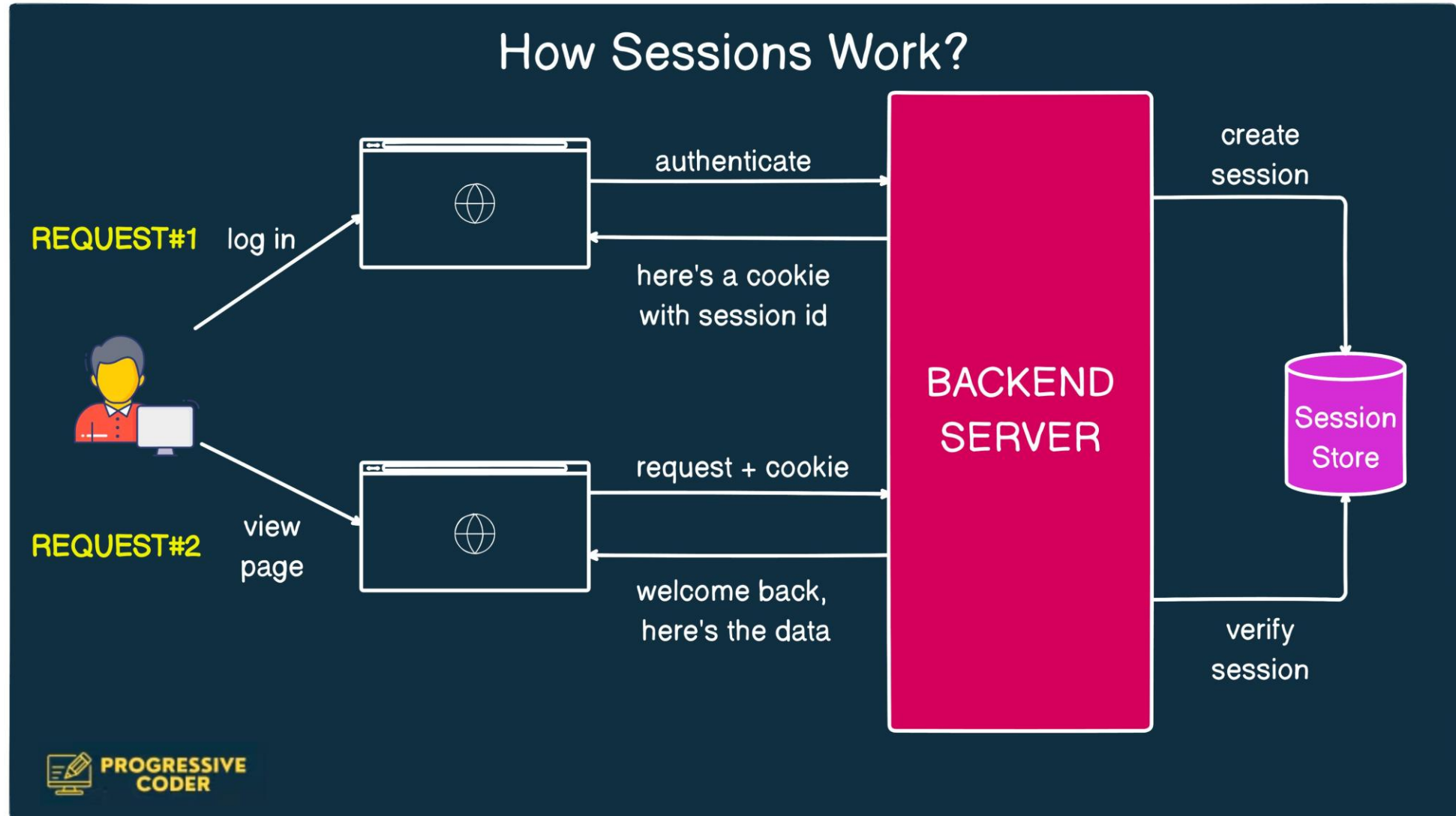
Como funciona o fluxo?

1. **Usuário envia suas credenciais** (login e senha) para o servidor.
2. O servidor **valida as credenciais**.
3. Se estiver tudo certo, o servidor:
 1. Cria uma **sessão** (objeto que armazena dados sobre o usuário).
 2. Gera um **ID de sessão** e envia esse ID de volta para o cliente dentro de um **cookie**.
4. A cada nova requisição, o navegador **envia automaticamente o cookie**, e o servidor **recupera a sessão** com base nesse ID.

Autenticação via Sessão



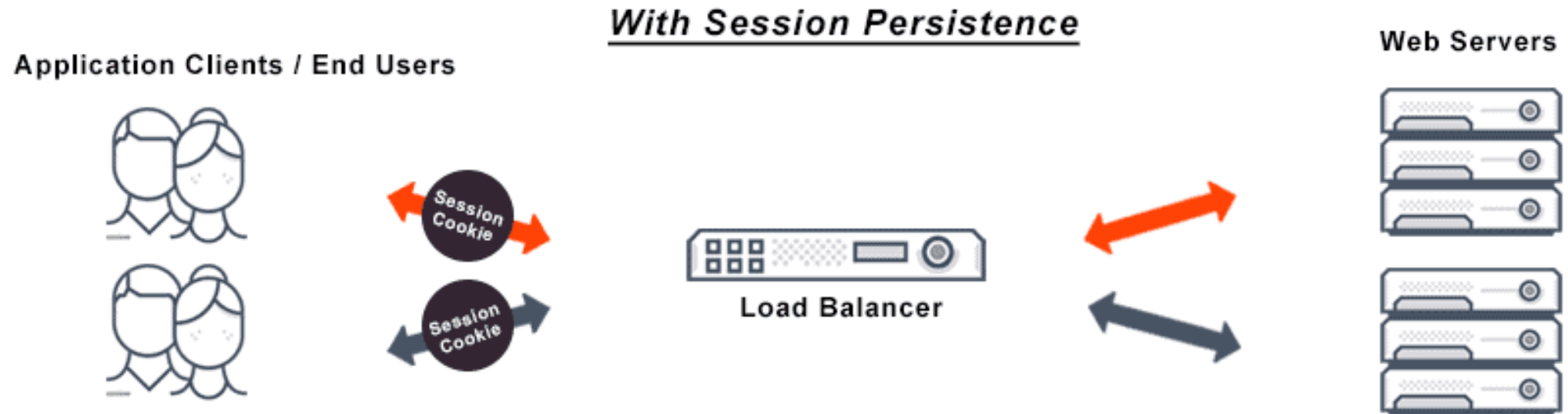
Autenticação via Sessão



O problema da autenticação via sessão

- O problema reside no fato de que as **informações de sessão são geralmente armazenadas na memória RAM de cada servidor** individualmente.
- Isso significa que uma sessão válida em um servidor específico pode não ser reconhecida como válida em outro.
- Imagine o seguinte cenário:
- um usuário faz login em um servidor A, estabelece uma sessão e recebe um cookie de autenticação.
- Se, por qualquer motivo, a próxima requisição dele for direcionada para o servidor B, as informações de sessão não serão compartilhadas automaticamente entre os servidores.
- Isso leva a uma situação desconfortável em que o usuário é tratado como não autenticado, mesmo estando logado em outro servidor do sistema distribuído.

Comparativo em sistemas distribuídos

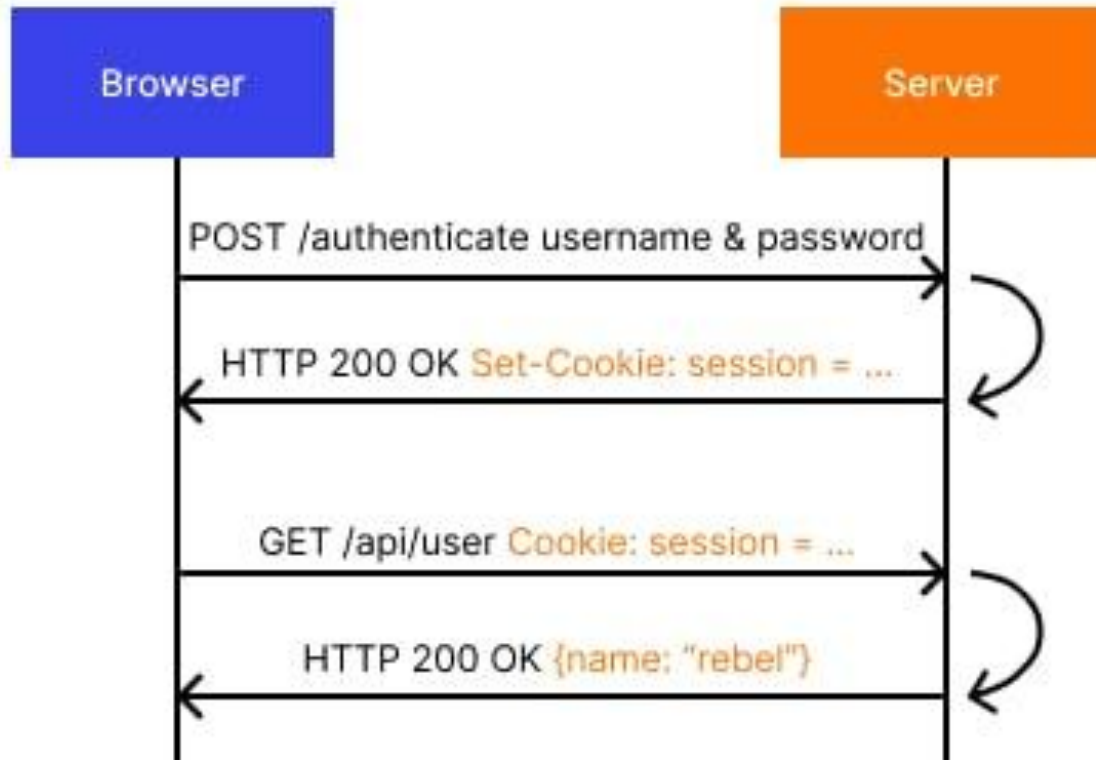


Autenticação via sessão

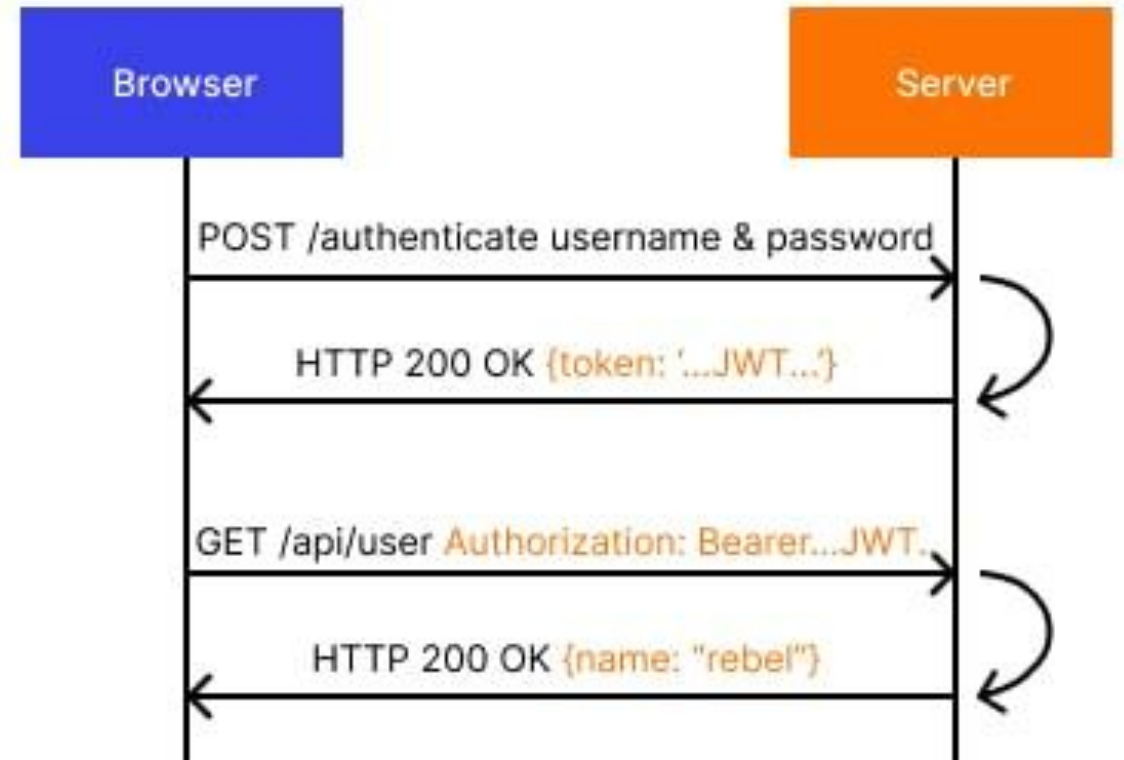
- Autenticação via token é um modelo muito utilizado em **APIs REST** modernas porque **mantém o servidor stateless** , ou seja, o servidor **não precisa guardar sessão** do usuário.
- **Como funciona?**
 - O cliente envia usuário e senha para o servidor (ex: /login).
 - O servidor valida as credenciais.
 - Se estiver tudo certo, o servidor gera um token de autenticação (ex: um JWT).
 - O cliente guarda esse token e envia em todas as requisições protegidas, geralmente no cabeçalho HTTP Authorization.

Autenticação – Sessão (Cookie) x Token


Traditional Cookie-based Authentication



Modern Token-based Authentication



Autenticação via sessão

-  O que é esse token?
- Normalmente, o token é um **JWT (JSON Web Token)**:
Um formato seguro e compacto que pode conter **dados codificados**, como:
 - ID do usuário
 - papel (role)
 - tempo de expiração
- O que o servidor faz com o token?
 - O servidor não guarda o token.
 - A cada requisição, ele apenas valida o token recebido, usando uma chave secreta ou chave pública, e decide se o acesso será permitido.

Autenticação via sessão

-  **Exemplo de Token (JWT Token)**

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoiYWRtaW4iOnRydWU9Im1hdCI6MTUxNjIzOTAyMn0.KMUFsIDTnFmyG3nMiGM6H9FNFUROf3wh7SmqJp-QV30

Autenticação Via Sessão

PRÓS

- Simplicidade: A autenticação via sessão é relativamente simples de implementar, pois o servidor mantém o controle do estado de autenticação para cada usuário.
- Controle de sessão: Com a autenticação via sessão, o servidor pode gerenciar o tempo de expiração da sessão e encerrar sessões ativas, proporcionando um maior controle sobre o acesso.

CONTRAS

- Estado do servidor: Para manter o estado da autenticação, o servidor precisa armazenar informações sobre cada sessão ativa, o que pode aumentar a complexidade e a necessidade de recursos.
- Escalabilidade: O armazenamento de informações de sessão pode se tornar um gargalo de escalabilidade à medida que o número de usuários e sessões aumenta.

Autenticação Via Token

PRÓS

- Stateless: A autenticação via token é stateless, o que significa que o servidor não precisa armazenar informações sobre o estado de autenticação. Isso aumenta a escalabilidade e a performance do servidor.
- Desacoplamento: Os tokens são autossuficientes e podem ser transmitidos por diferentes sistemas ou domínios, permitindo uma arquitetura mais distribuída.
- Suporte a APIs RESTful: A autenticação via token é bem adequada para APIs RESTful, onde a natureza stateless é um requisito importante.

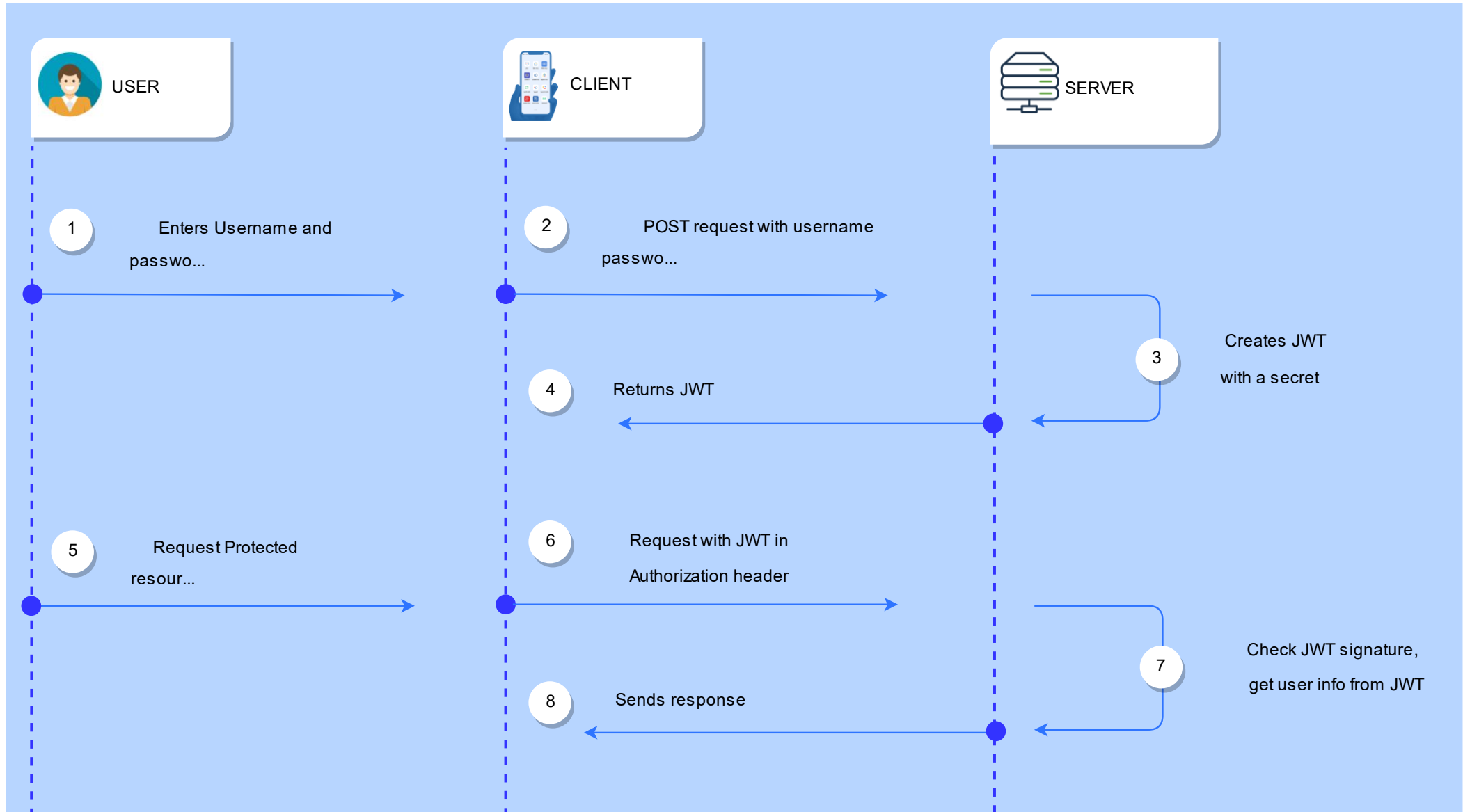
CONTRAS

- Complexidade de implementação: Comparada à autenticação via sessão, a autenticação via token pode ser mais complexa de implementar, exigindo a definição de fluxos de autenticação e a escolha de algoritmos de assinatura adequados.
- Gerenciamento de tokens: É necessário implementar mecanismos para gerar, armazenar e revogar tokens, bem como garantir a segurança da chave de assinatura.
- Limitações de revogação: Se um token válido for comprometido ou precisar ser revogado antes de sua expiração, é necessário implementar um mecanismo adicional para lidar com isso.

Conclusão

- Em resumo, a autenticação via sessão é mais simples de implementar e é adequada para cenários em que o controle de sessões e requisitos complexos de autenticação são necessários. Por outro lado, a autenticação via token é stateless, escalável e mais adequada para arquiteturas distribuídas e APIs RESTful. A escolha entre as duas abordagens depende das necessidades específicas do sistema e dos requisitos de segurança.

JSON Web Token

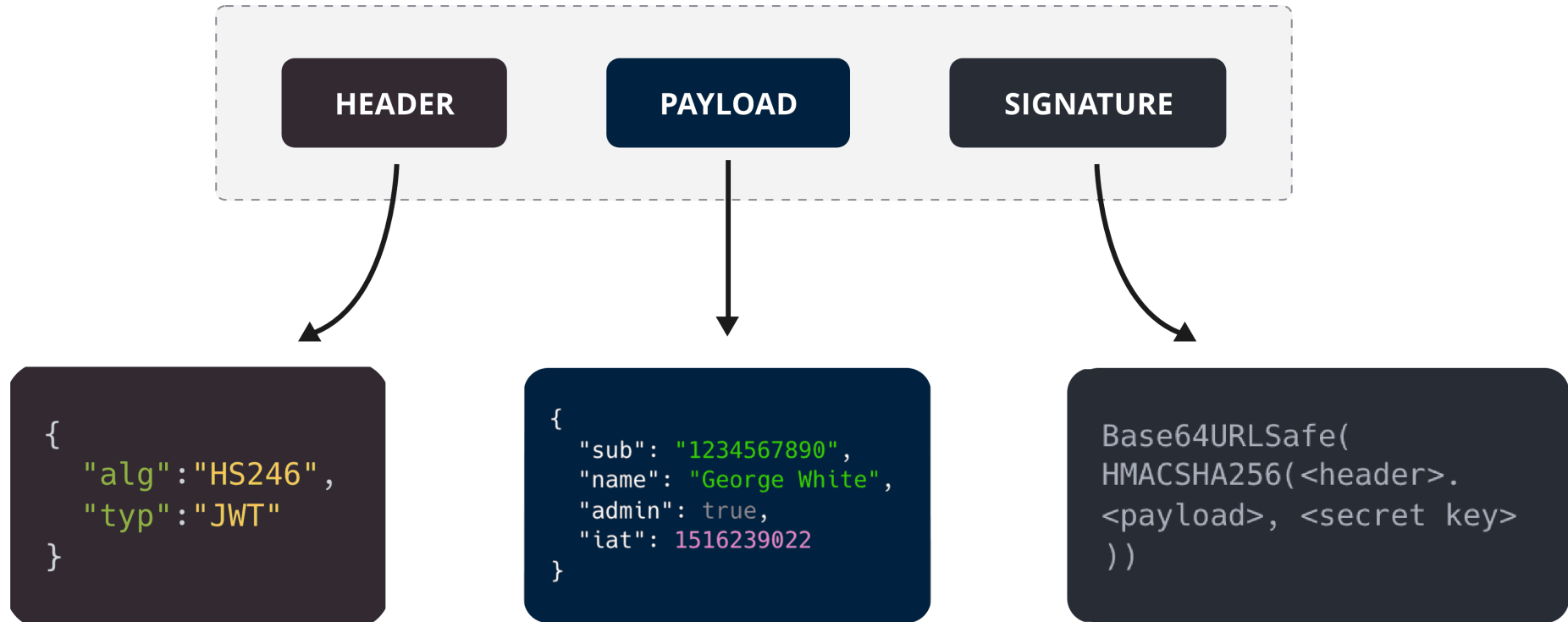


JSON Web Token - JWT

- JSON Web Tokens (JWT) é um formato compacto e seguro para representar informações entre duas partes de forma digitalmente assinada. É frequentemente utilizado para autenticação em APIs REST devido à sua simplicidade e eficiência.
- Um JWT é composto por três partes: o cabeçalho (header), o payload e a assinatura (signature). O cabeçalho especifica o tipo do token e o algoritmo de assinatura utilizado. O payload contém as informações (claims) que podem incluir dados do usuário, permissões ou qualquer outra informação relevante. A assinatura é gerada usando a chave secreta do servidor, que pode ser usada para verificar a integridade do token.

JSON Web Token - JWT

Structure of a JSON Web Token (JWT)



Esquema de autenticação Bearer

- O termo "Bearer" é usado no contexto de autenticação para indicar o tipo de esquema de autenticação utilizado para transmitir um token de acesso em uma solicitação HTTP. Ele não tem um significado específico além de identificar o tipo de esquema de autenticação sendo utilizado.
- O uso do termo "Bearer" surgiu no contexto do protocolo OAuth 2.0, que é um protocolo de autorização amplamente utilizado para autenticação em APIs. No OAuth 2.0, o esquema "Bearer" é utilizado para transmitir tokens de acesso, que são usados para autenticar e autorizar as solicitações.
- O termo "Bearer" refere-se ao fato de que o token de acesso é portado (bearer) pelo cliente e usado como uma credencial para provar a autenticação em cada solicitação subsequente. Em outras palavras, o cliente "porta" o token e o apresenta ao servidor em cada requisição para acessar recursos protegidos.

Esquema de autenticação Bearer

- Ao utilizar o esquema "Bearer", o token de acesso é incluído no cabeçalho "Authorization" da solicitação HTTP da seguinte maneira:

```
Authorization: Bearer <token>
```

- O termo *<token>* é substituído pelo valor do JWT, que é um longo conjunto de caracteres alfanuméricos. Ao receber uma solicitação com o cabeçalho "Authorization" contendo o token JWT, o servidor pode validar a autenticidade do token e, em seguida, autorizar ou negar o acesso aos recursos solicitados com base nas informações contidas no payload do JWT.

Esquema de autenticação Bearer

- O uso do JWT com o esquema "Bearer" traz algumas vantagens significativas.
 - Primeiro, como o token é autossuficiente e contém as informações necessárias, o servidor não precisa consultar um banco de dados ou manter estado para verificar a autenticidade do token. Isso permite uma escalabilidade e desempenho melhores.
 - Além disso, como o token é assinado digitalmente, qualquer alteração no token invalidará a assinatura, garantindo a integridade dos dados.
- No entanto, é importante mencionar que a segurança do JWT depende da proteção adequada da chave secreta usada para assinar os tokens. Se um token JWT for comprometido ou a chave secreta for exposta, um atacante pode falsificar tokens e ganhar acesso não autorizado aos recursos protegidos.

Conclusão

- Em resumo, o JWT é um formato popular para autenticação em APIs REST, e o esquema "Bearer" é um tipo de esquema de autenticação utilizado para transmitir tokens de acesso em uma solicitação HTTP, comumente usado para transmitir tokens JWT.
- Essa combinação oferece uma maneira eficiente e segura de autenticar e autorizar usuários em APIs REST, permitindo o acesso controlado aos recursos protegidos.

Spring Security



Spring Security

- O Spring Security é um poderoso framework de segurança Java, desenvolvido pelo Spring Framework. Ele oferece uma ampla gama de recursos e funcionalidades para lidar com autenticação e autorização em aplicativos web e API REST.
- O objetivo principal do Spring Security é fornecer uma camada de segurança robusta e flexível, facilitando a proteção dos recursos e dados sensíveis de um aplicativo. Ele simplifica a implementação de requisitos de segurança comuns, como autenticação baseada em formulário, autenticação via token, autorização baseada em papéis e muito mais.

Spring Security

- Autenticação: O Spring Security oferece suporte a vários métodos de autenticação, incluindo autenticação baseada em formulário, autenticação via token (como JWT) e autenticação baseada em provedores externos (como OAuth). Ele também possui recursos para o gerenciamento de sessões, recuperação de senhas e proteção contra ataques de força bruta.
- Autorização: O Spring Security permite a configuração flexível de regras de autorização para controlar o acesso aos recursos. Ele suporta a autorização baseada em papéis (role-based), onde as permissões são concedidas com base em funções ou papéis atribuídos aos usuários. Além disso, o Spring Security oferece suporte a expressões de segurança, que permitem a criação de regras mais granulares para autorização.

Spring Security

- Proteção contra ataques: O Spring Security inclui medidas de proteção contra uma variedade de ataques comuns, como cross-site request forgery (CSRF), cross-site scripting (XSS) e injeção de SQL. Ele implementa filtros de segurança para prevenir esses ataques e fornece recursos para configurar políticas de segurança personalizadas.
- Integração com o ecossistema Spring: O Spring Security é altamente integrado com outros componentes do ecossistema Spring, como o Spring MVC e o Spring Boot. Ele se integra perfeitamente com esses componentes, permitindo uma configuração fácil e coesa da segurança em aplicativos Spring.

Spring Security

- Extensibilidade: O Spring Security é altamente extensível, permitindo que os desenvolvedores personalizem e estendam seu comportamento para atender às necessidades específicas do aplicativo. Ele fornece pontos de extensão e hooks para adicionar funcionalidades personalizadas, como autenticação de dois fatores, integração com provedores externos e personalização do fluxo de autenticação.

Conclusão

- Em resumo, o Spring Security é um framework robusto e flexível para adicionar recursos de autenticação e autorização a aplicativos Java. Ele simplifica a implementação de requisitos de segurança comuns e fornece recursos abrangentes para proteger os recursos e os dados sensíveis do aplicativo. Com sua integração perfeita com o ecossistema Spring, ele se torna uma escolha popular para a segurança de aplicativos baseados em Java.