Aula 18 - Cookies, sessão e armazenamento

**biscoitos**

**A necessidade de conhecer as informações do cliente**

Quando desenvolvemos um site, temos que contemplar que a forma como um cliente interage geralmente é diferente, por isso é importante ter algum recurso para saber informações sobre determinados detalhes de informações e comportamento de um cliente, para que o servidor possa usar isso em seu próprio critério.

Como podemos acompanhar os clientes do nosso site e conseguir obter um pouco mais de informação de contacto e/ou comportamental dos clientes que nos visitam? A resposta: biscoitos.

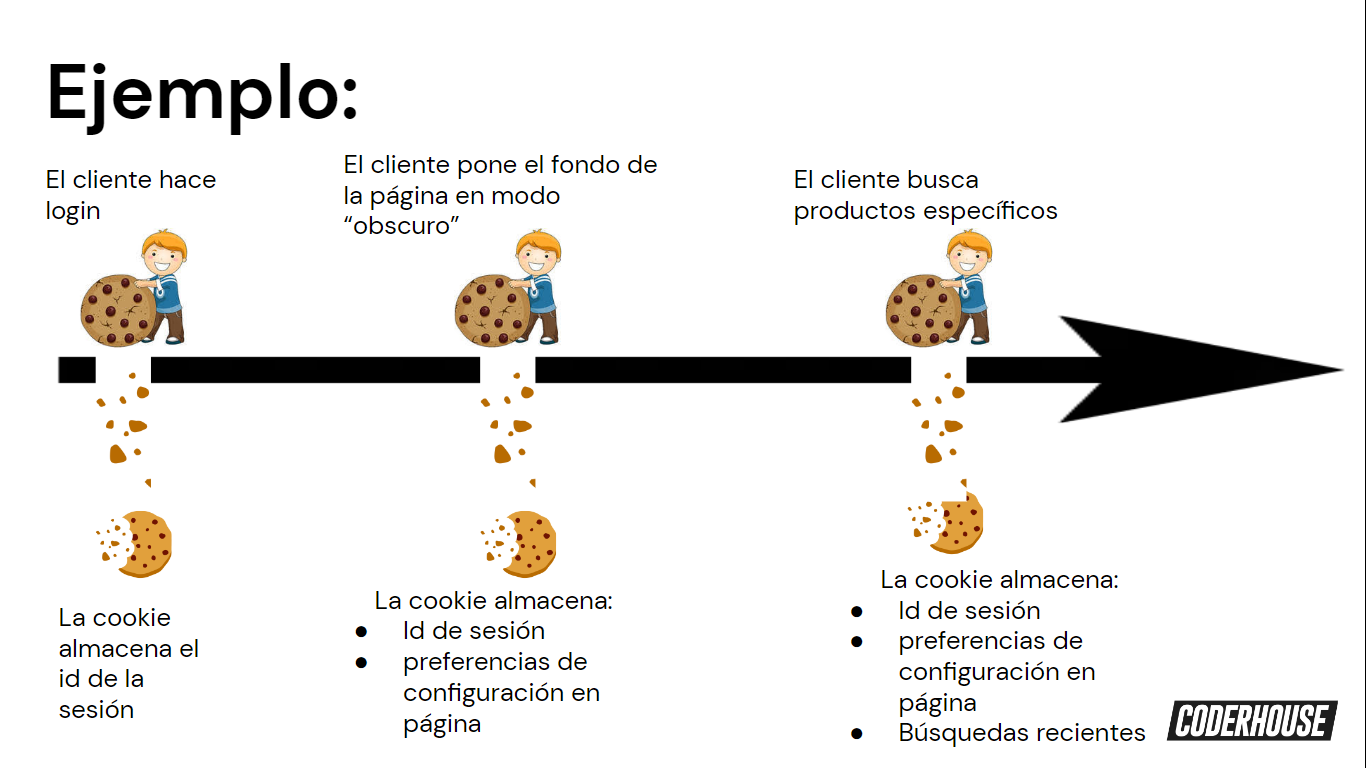
**O que é um biscoito?**

Um cookie é um arquivo de texto muito pequeno onde podemos armazenar informações dentro do navegador, para que ele possa viajar entre as solicitações e servir como um recipiente leve de informações necessárias para poder processar determinadas solicitações.

Alguns dos dados que geralmente são armazenados em um cookie são:

* nomes de usuários
* IDs de sessão (que abordaremos mais adiante)
* Preferências de navegação para sua página.

**Rastros que um usuário costuma deixar ao navegar na web**



Importante!

Os cookies residem no navegador, portanto, são facilmente acessíveis por vários elementos externos.

Não armazenamos informações confidenciais em um cookie por nenhum motivo. Nunca guardamos informações sobre formas de pagamento, senhas ou quaisquer dados que possam comprometer a segurança do cliente.

**Características**

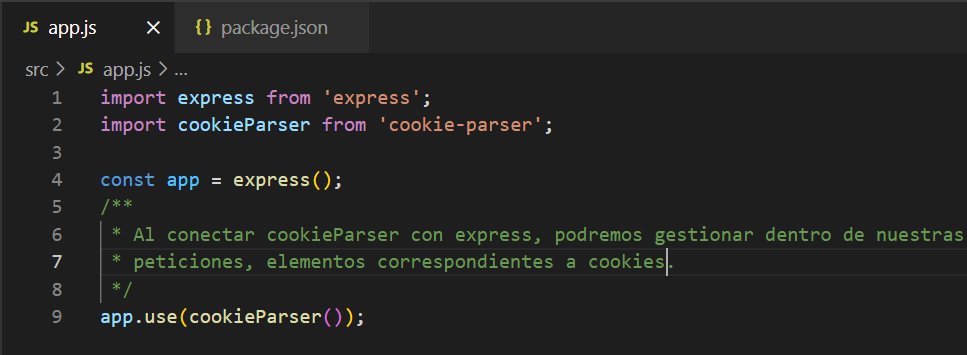
* Os cookies podem ser configurados para toda a vida. Depois de concluído, o cookie é excluído do navegador.
* Quando armazenados no lado do cliente, o espaço disponível é limitado, por isso é recomendável escolher adequadamente o que será salvo como cookie.
* Podemos atribuir-lhes chaves secretas para aumentar a segurança
* Eles vivem no navegador, então não armazenamos dados confidenciais

**Começando a usar cookies**

Começamos instalando o express e o módulo cookie-parser



Posteriormente, seguindo a arquitetura que fizemos em projetos anteriores, iremos configurar nosso servidor. Usaremos o middleware com app.use



**Uso de cookies: definir, obter e limpar**

**Defina um cookie.**

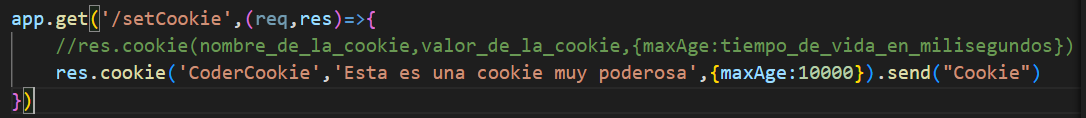
Um cookie deve ser definido dentro do fluxo de vida de uma solicitação, portanto, chamaremos um endpoint chamado /setCookie onde usaremos o objeto res para atribuir um cookie ao cliente em seu navegador.

Para ler o cookie definido, usaremos o objeto req no endpoint /getCookies, pois, como o cliente possui o cookie em seu navegador, ele deve enviá-lo por meio do referido objeto.

Além disso, também chamaremos um endpoint chamado /deleteCookie onde usaremos o objeto res, a fim de limpar o cookie atribuído ao cliente em seu navegador.

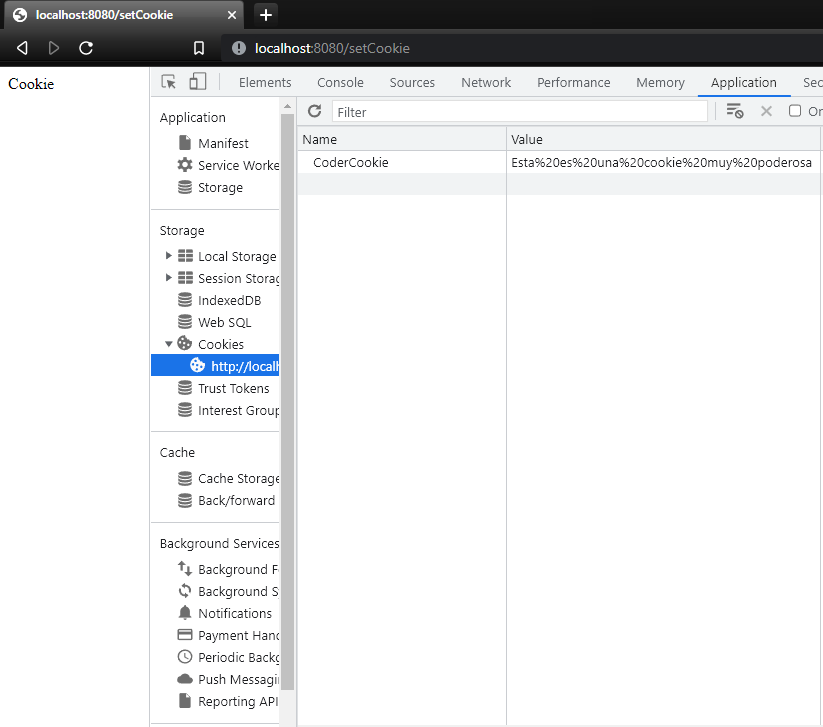
Exemplo: definir um cookie

Usando o objeto res para enviar o cookie para o cliente.



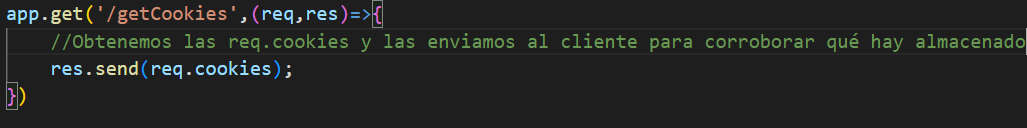
Se não colocarmos o parâmetro maxAge, o cookie persistirá até que seja deletado (sem tempo de vida definido).

Verificando se o cookie está definido na guia "Aplicativo"



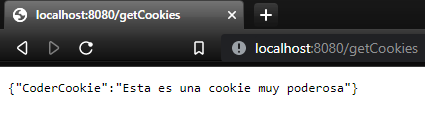
Exemplo: pegar um biscoito

Usando o objeto req para verificar os cookies



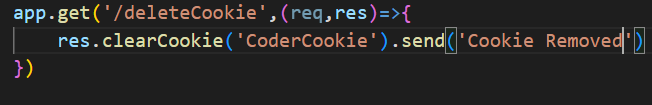
Neste caso, enviamos todos os cookies para o cliente. Se quisermos acessar um cookie específico, podemos chamá-lo de req.cookies.cookie\_name

Obtemos o cookie recém-configurado



Exemplo: Excluir um cookie

Usando o objeto res para excluir o cookie que havíamos definido



Se o cookie já foi excluído ou expirou devido à expiração, não há problema. O clearCookie passa a ignorá-lo.

**Adicionando segurança ao cookie: Cookies assinados**

O que é "assinar" um cookie?

Como os cookies são armazenados no navegador, eles podem ser alterados com muito mais facilidade do que se residissem no servidor. É por isso que precisamos adicionar um fator de segurança para que o cookie seja "invalidado" caso tenha sido modificado.

Não podemos impedir que alguém externo altere o cookie, mas podemos indicar que, caso o cookie não seja mais exatamente idêntico ao gerado, ele o passa como um cookie inválido.

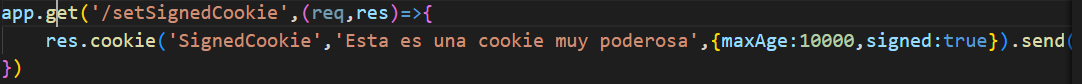
Podemos usar o mesmo cookieParser

Se inicializarmos:



Não há necessidade de instalar nada novo, apenas configuraremos a inicialização do cookieParser. Isso será alcançado adicionando um segredo no momento da inicialização

Poderemos assinar os cookies para maior segurança com base na lógica proposta, basta colocar um {signed:true} na definição do cookie:



Sobre cookies assinados

* Para acessar um cookie assinado, eles não estarão mais disponíveis em req.cookies, mas em req.signedCookies, portanto, você deve pensar bem quais cookies corresponderão a qual lado
* Se tentarmos acessar um cookie assinado que foi alterado por algum motivo, tentar acessá-lo retornará apenas false.

**Dando identidade ao cliente: sessões**

Vamos voltar a um conceito interessante: a conexão sem estado.

Como sabemos, uma das características da nossa API REST é a conexão stateless, lembre-se que isso significa que o servidor recebe uma requisição do cliente e retorna uma resposta... assim, sem contexto prévio.

O cliente não sabe de onde obtém a informação que está solicitando, e o servidor não se importa com o que o cliente fará com a informação que acabou de entregar.

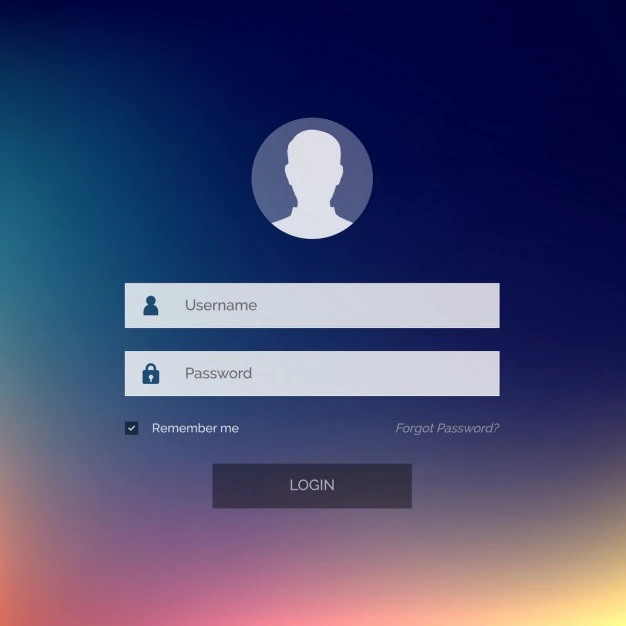
Então, como o servidor sabe sobre o usuário ao fazer uma solicitação?

Isso certamente levanta a questão para nós: como eles sabem quem eu sou em um site? Como todo o fluxo de uma compra seria gerenciado se meu servidor funcionasse sem estado?

Para resolver essas situações, o servidor deve sempre assumir a identidade do cliente que está fazendo a requisição. Ou seja, o cliente alimenta o servidor a cada requisição com as informações que ele precisa processar. O servidor não armazena nada para si mesmo.

Atualmente, o cliente deve nos enviar tais informações de consultas, parâmetros, corpo e cookies. Tudo isso enviado da frente. E se delegarmos um pouco mais de responsabilidade nas costas? Vamos gerenciar um sistema de sessão.

Usuário, identifique-se!



O sistema de sessão permitirá que o servidor armazene informações sobre o cliente, para que este permaneça identificado ao fazer solicitações.

Finalmente quebramos o anonimato! Depois que o cliente passa por um processo de login, podemos processar essa informação para manter o cliente reconhecido e poder fornecer respostas específicas de acordo com sua função na página.

**Usando sessão**

Instalamos o módulo com npm

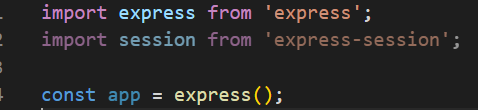
A sessão é um módulo de nó projetado para lidar com o sistema de sessão definido anteriormente



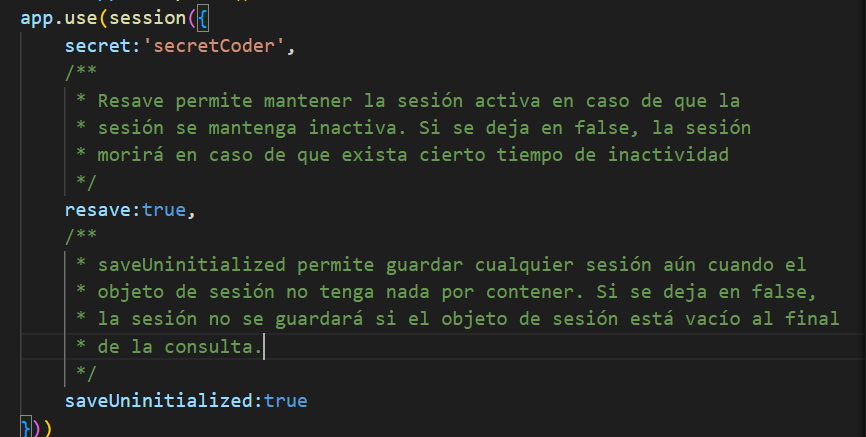
Começamos instalando a sessão em nosso projeto



E vamos inicializar nosso servidor importando a sessão



Por fim, colocamos a sessão como middleware para nosso servidor expresso

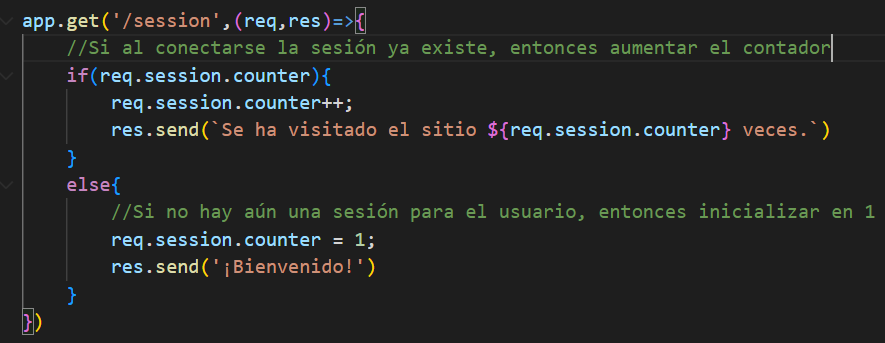


Fazendo logoff no endpoint

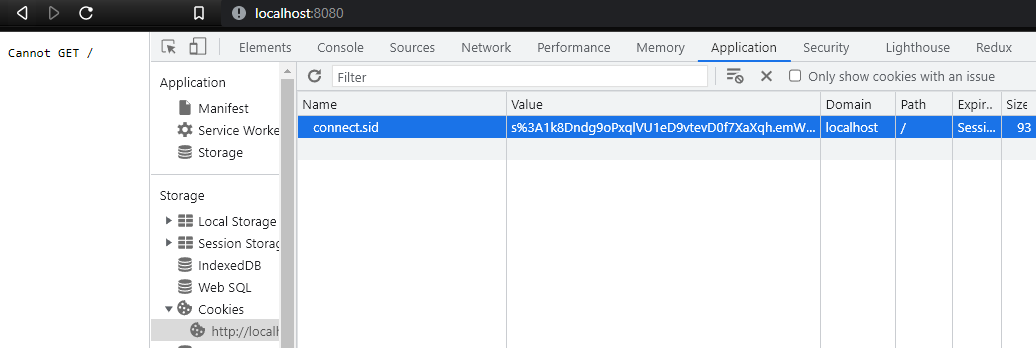
no ponto final

"/sessão"

Vamos inicializar a sessão. Se o contador não existir, ele o inicializará em 1 e, se existir, o aumentará para manter a contagem de visitas.



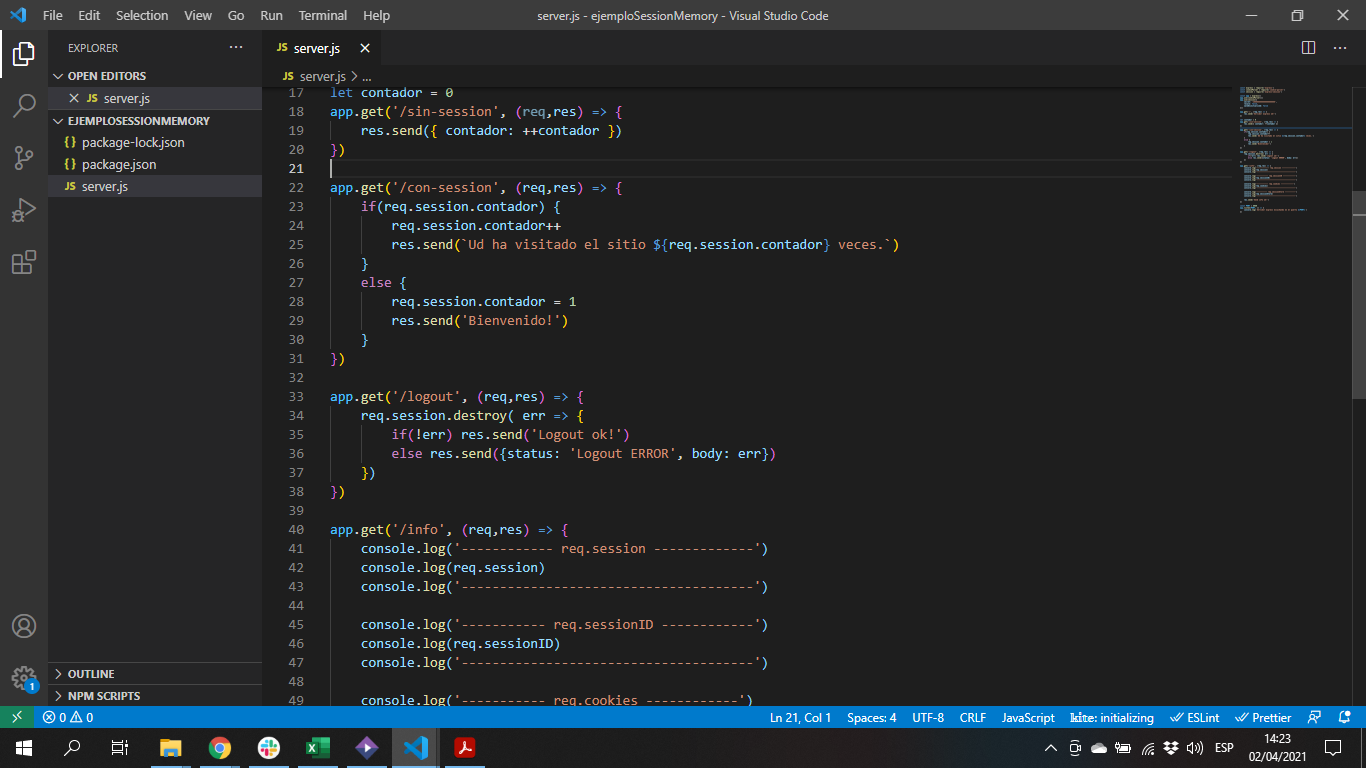
Notamos que, apesar de não ter chamado o endpoint, a sessão já inseriu um ID de sessão com o nome de connect.sid





**Excluir dados da sessão**

Para remover dados de uma variável de sessão, use o parâmetro request e o método destroy. Um retorno de chamada é passado como um parâmetro.

****

**Entrar com sessão**

app . get ( '/login' , ( req , res ) => {

const { nome de usuário , senha } = req . consulta

if ( usuário !== 'pepe' || senha !== 'pepepass' ) {

retornar res . enviar ( 'falha no login' )

}

req . sessão . nome de usuário = nome de usuário

req . sessão . administrador = verdadeiro

res . send ( 'login com sucesso!' )

})

**Middleware de autenticação**

função auth ( req , res , próximo ) {

if ( req . sessão ?. user === 'pepe' && req . sessão ?. admin ) {

retornar próximo ()

}

retornar res . estado ( 401 ). send ( 'erro de autorização!' )

}

Através desses middlewares, o acesso a determinadas rotas pode ser limitado àqueles usuários que são administradores (ou, por exemplo, outros a qualquer usuário logado).

Se o usuário salvo na sessão corresponder e também for admin, segue a rota, caso contrário, retorna um erro.

**aplicativo de middleware**

app . get ( '/private' , auth , ( req , res ) => {

res . send ( 'se você está vendo isso, você já está logado!' )

})

Ao aplicar o middleware auth à rota /content, ele ficará acessível somente após o login do usuário.

Além disso, dependendo do código do middleware, um determinado usuário ou um determinado tipo de usuário (admin ou usuário comum, por exemplo) pode ser especificado.

**Sair com sessão**

app . get ( '/logout' , ( req , res ) => {

req . sessão . destruir ( err => {

if ( err ) {

retorno res . json ({ estado: 'Logout ERROR' , corpo: erro })

}

res . send ( 'Sair ok!' )

})

})

resumo da aula

* Conceito de cookies e sessões
* Aplicação de cookies e sessões
* sistema de login elementar