Angular

Qinshift Academy

Authentication

Qinshift Academy

What is Angular Authentication?

An Angular application's authentication procedure verifies users' identities. It is a basic security mechanism that restricts program resources and operations to authorized users.

Angular applications authenticate users by validating usernames and passwords and giving them access.

Why Does Authentication Matter?

Authentication matters for many reasons:

Data Security:

Application authentication protects sensitive data and functions against unauthorized users. It protects user-specific data from malicious use.

User Privacy:

By confirming user identities, authentication protects privacy. It restricts access to personal data, communication, and transactions to authorized users.

Regulatory Compliance:

User data protection is strictly regulated in several businesses and areas. Strong authentication helps organizations comply with regulatory obligations and avoid penalties.

Preventing unauthorized access:

Unauthorized access can cause data breaches, identity theft, and other cybercrimes. Identity authentication prevents unauthorized users from exploiting weaknesses and accessing sensitive data.

User Accountability:

Authentication links application user behaviors to individuals. Auditing, tracking user activity, and settling disputes require accountability.

Enhancing User Trust:

Data-secure apps are more trusted. Trust in authentication boosts user confidence and retention.

Impersonation Prevention:

Authentication prevents malevolent users from impersonating legitimate users. Applications can prevent fraud by authenticating user IDs.

JWT Angular Authentication

Pros of JWT for Angular Authentication:

- Stateless: JWTs hold no session data, decreasing server overhead.
- Decentralized: Token verification without database queries improves scalability.
- Payload Flexibility: The JWT payload can store bespoke data, giving user responsibilities and permissions flexibility.
- Cross-Domain Authentication: Allows smooth integration between domains or services.

Cons of JWT for Angular Authentication:

- Token Size: substantial JWTs can slow network performance if payload data is substantial.
- Limited Revocation: JWTs cannot be canceled sooner without complications.
- Security Dependency: The token signature secret key is crucial to security.

The Building Blocks of JWT: Header, Payload, Signature

JWTs are made up of three parts: a header, a payload, and a signature. Here's what each part does:

1. **Header**: This part tells us what kind of token we're dealing with (which is JWT) and the algorithm used for creating the signature.
2. **Payload**: This is where the user data is stored. It's also where you'll find details about when the token was issued and when it will expire.
3. **Signature**: Think of the signature as a safety seal. It's created by taking the encoded header, the encoded payload, and a secret key, then running them through a special algorithm. The resulting signature helps confirm that the token hasn't been tampered with.

These three parts work together to create a JWT, which is a vital tool in ensuring user data stays secure and website interactions remain smooth.

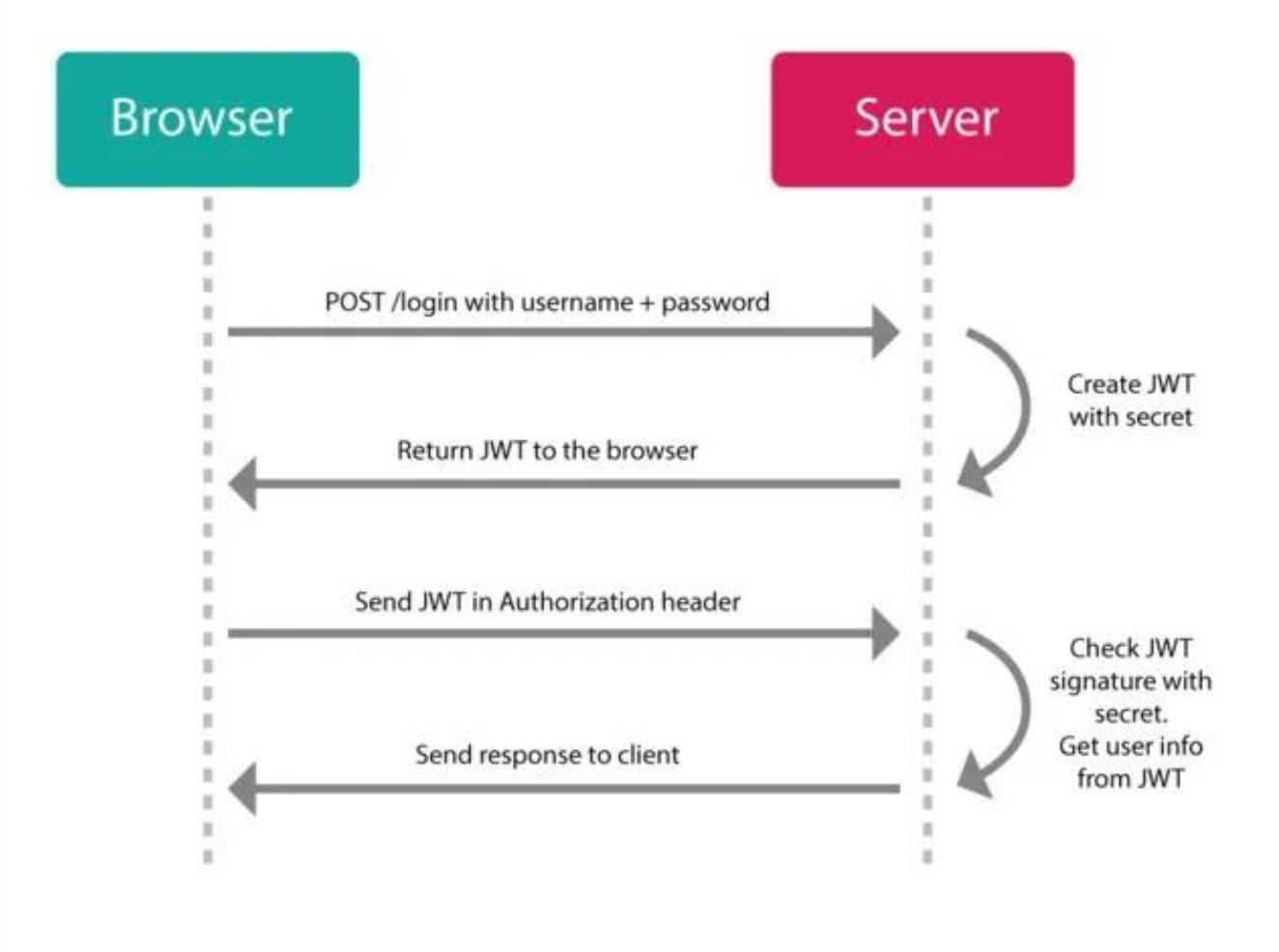
How JWT Authentication Flow Works

Let’s look at how JWT actually works in practice step by step:

- **Login**: You enter your username and password on the website. If the details are correct, the server generates a JWT and sends it back to you.
- **Store and Send**: Your browser then stores this token (often in a place called local storage). Every time you visit a new page on the site or request a protected resource, your browser automatically sends the JWT along in the headers of the HTTP request.
- **Verify and Allow**: The server gets this request, and the JWT with it. It checks the JWT’s signature to make sure it’s legit and hasn’t been messed with. If everything checks out, the server provides the resources you asked for.

How JWT Authentication Flow Works

In short, JWT helps websites remember who's logged in and keeps your session active as you move around the site.



Implementing JWT in Angular using Interceptors

What are Interceptors?

In Angular, Interceptors are like having checkpoints where changes can be made, like adding headers to a request or logging responses from a server. In our case, we'll use an interceptor to automatically add our JWT to every request that needs it.

Getting the Tools: Installing Necessary Packages

To start working with JWTs in Angular, we'll need to install a couple of packages. These are some extra tools that Angular needs to handle JWTs. They're called `@auth0/angular-jwt` and `@angular/common/http`. We can get these using a tool called npm (Node Package Manager) by typing the following in our project directory:

Making an Interceptor: Injecting the JWT into HTTP Requests

Next, we'll create an interceptor. Its job is to take the JWT we stored and add it to the header of any HTTP requests that need it. Here's a simple example:

```
@Injectable()
export class JwtInterceptor implements HttpInterceptor
{
  intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>>
  {
    let token = localStorage.getItem('access_token');
    if (token) {
      request = request.clone({
        setHeaders: {
          Authorization: `Bearer ${token}`
        }
      });
    }
    return next.handle(request);
  }
 }
```

This interceptor checks if we have a token in local storage. If we do, it adds that token to the HTTP request's 'Authorization' header.

Registering the HTTP Interceptor in the AppModule

Next, we will register this interceptor in the app module so that the token will be added to all outgoing HTTP requests:

```
import { HttpClientModule, HTTP_INTERCEPTORS } from '@angular/common/http';
import { MyInterceptor } from './path-to-your-interceptor-file';

@NgModule({
  imports: [HttpClientModule],
  providers: [
    {
      provide: HTTP_INTERCEPTORS,
      useClass: MyInterceptor,
      multi: true
    }
  ]
})
export class AppModule { }
```

Staying Fresh: Handling Token Expiration and Refresh

JWTs often have an expiration time for security reasons. But what happens when a token expires? Usually, the server will return a 401 Unauthorized status. When we get that, we can ask the server to give us a new JWT (this is called “refreshing” the token).

Handling this refresh depends a lot on how your server is set up. But the key point is to remember that JWTs can expire, and you’ll need a strategy in place to either refresh the token or prompt the user to log in again.

Security Considerations with JWT and Angular

Potential Vulnerabilities

While JWT and Angular can provide a secure framework for authentication, they're not without potential security concerns.

1. **Token Theft**: If a JWT is stolen, someone else could use it to impersonate the user. This could be done through cross-site scripting (XSS) attacks where malicious scripts are injected into trusted websites.
2. **Weak Secret Key**: JWTs depend on a secret key for signing. If this key is not strong enough or gets compromised, it could allow an attacker to create their own valid tokens.
3. **Token Storage**: JWTs are often stored in local storage, which is vulnerable to XSS attacks. An attacker could potentially retrieve the token from local storage and use it.

Best Practices for Secure Use of JWT

Despite these potential vulnerabilities, there are ways to use JWT securely:

1. **Use HTTPS**: Always use HTTPS for transferring JWTs between client and server. This encrypts the communication, making it harder for attackers to steal tokens.
2. **Strong Secret Key**: Use a strong and unique secret key for signing JWTs. Never disclose this key.
3. **Token Expiration**: Keep the expiration time of JWTs as short as possible. This reduces the window of time an attacker could use a stolen token.
4. **Store Tokens Securely**: Consider alternatives to local storage for token storage, like HTTP-only cookies. These cannot be accessed by JavaScript, which can prevent XSS attacks.
5. **Handle Expiration**: Always check the expiration of JWTs and handle expired tokens appropriately. Refresh tokens when necessary but do so securely.

By being aware of these potential vulnerabilities and following best practices, you can ensure a more secure implementation of JWT in Angular.

Questions?

Trainer Name

Trainer

trainer@mail.com

Assistant Name

Assistant

assistant@mail.com

Qinshift Academy