

Token-based Authentication in Angular 6 with ASP.NET Core 2.1

13 Aug 2025

73.2K Views

Token-Based Authentication in Angular 6 with ASP.NET Core 2.1: An Overview

Hello there! Have you ever wondered how [Angular 6](#) and [ASP.NET Core 2.1](#) work together utilizing Token-based Authentication? In this [Angular tutorial](#),

Read More: [Top 50+ Angular Interview Questions & Answers](#)

What are JWTs?

JWTs are signed tokens that contain user data and claims. They are small, tamper-proof, or stateless, making them suitable for secure client-server communication.

The screenshot shows the jwt.io website interface. At the top, there's a navigation bar with links for Debugger, Libraries, Introduction, Ask, Get a T-shirt!, and Create. Below the header, there are two main sections: "Encoded" on the left and "Decoded" on the right.

Encoded: A text input field labeled "PASTE A TOKEN HERE" contains the following JWT string:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXV
CJ9.eyJzdWIiOiJKaWduZXNoIFRyaXZlZ
GkiLCJ1bWFpbCI6InRlc3QuYnRlc3RAZ2
1haWwuY29tIiwiRGF0ZU9mSm9pbmcioiI
wMDAxLTAxLTAxIiwianRpIjoiYzJkNTZj
NzQtZTc3Yy00ZmUxLTgyYzAtMzlhYjhmn
zFmYzUzIiwiZXhwIjoxNTMyMzU2NjY5LC
Jpc3Mi0iJUZXN0LmNvbSIsImF1ZCI6I1R
lc3QuY29tIn0.8hwQ3H9V8mdNYrFZSjbC
pWSyR1CNyDYHcGf6GqqCGnY|
```

Decoded: This section shows the decoded components of the JWT.

- HEADER: ALGORITHM & TOKEN TYPE**

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```
- PAYOUT: DATA**

```
{
  "sub": "Jignesh Trivedi",
  "email": "test.btest@gmail.com",
  "DateOfJoining": "0001-01-01",
  "jti": "c2d56c74-e77c-4fe1-82c0-
39ab8f71fc53",
  "exp": 1532356669, Mon Jul 23 2018 20:07:49 GMT+0
  "iss": "Test.com",
  "aud": "Test.com"
}
```
- VERIFY SIGNATURE**

HMACSHA256(

Why Use JWT (JSON Web Token)?



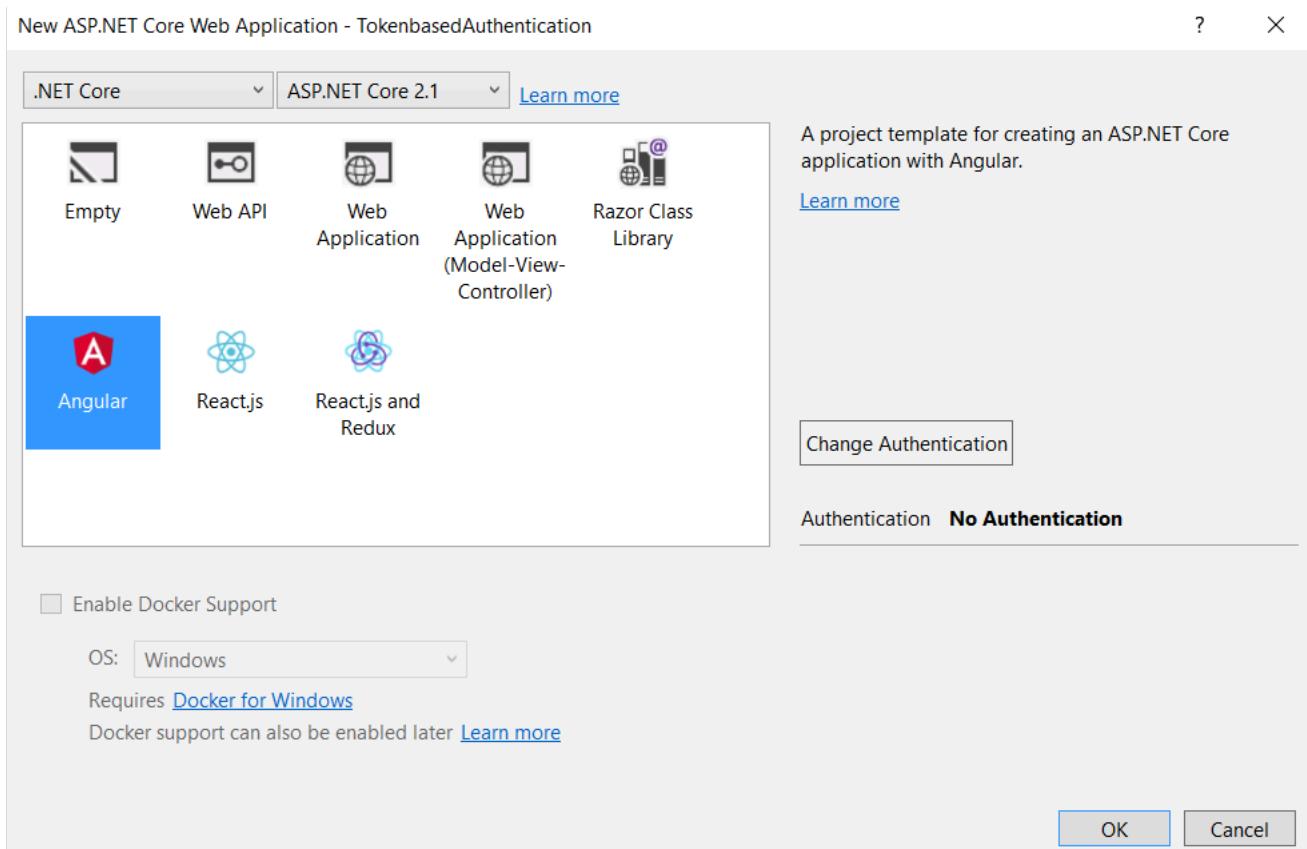
- Enable cross-application single sign-on.
- Small and tamper-resistant for effective communication.
- Modern online applications require scalability and dependability.

Softwares

The following software must be installed in our system before starting the work.

- ASP.NET Core 2.1 or above
- **TypeScript** 2.5 or above
- The latest version of **Node.js** and npm (node package manager)
- Editor (e.g., VS 2017, VS Code)

ASP.NET Core comes with many built-in templates such as **Angular**, **React**, etc. I am using the Angular template to demonstrate the concept. We can do this by creating a new ASP.NET Core Web Application.



Following the .Net Core CLI command will create an ASP.NET **Web API** project with Angular with the project name "TokenbasedAuthentication" in the current directory.

```
<dotnet new angular -n TokenbasedAuthentication
```

In this template, the Angular client App and Web API are shipped together in one project, however, we can create two separate projects:

1. **Angular**
2. **Web API**



Live Batches

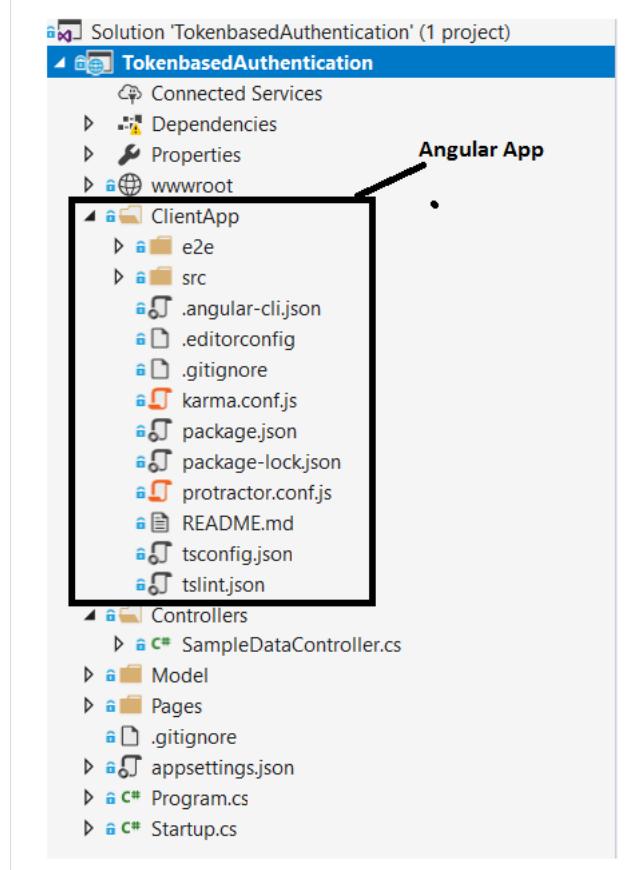


Masterclasses



Free Courses

Account



Configure JWT Token-based Authentication

Setting up a secure authentication system in which users log in to your Angular 6 application and then get a JSON Web Token (JWT) from your ASP.NET Core API.

Configure JWT Authentication in Startup.cs

- **AddAuthentication:** Use AddAuthentication to register the JWT authentication scheme.
- **Configure JwtBearer:** Use AddJwtBearer to specify options such as issuer, audience, and signing key.
- **Authorization:** In AddAuthorization, define authorization policies based on claims or responsibilities.

Example of Configure JWT Authentication in Startup.cs

```
services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(options =>
{
    options.TokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuer = true,
        ValidIssuer = "YourIssuerName",
        ValidateAudience = true,
        ValidAudience = "YourAudienceName",
        ValidateLifetime = true,
        IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("YourSecretKey"))
    };
});

services.AddAuthorization(options =>
```

```
options.AddPolicy("Authenticated", policy =>
    policy.RequireClaim("UserId"));
```

This code ensures that only users who possess valid JWT tokens issued by "YourIssuerName" & containing the "UserId" claim can access restricted endpoints.

Create Login Controller

Create a controller endpoint to receive and validate user credentials. Generate a JWT token using the configured values after successful login.

Example of Create Login Controller

```
[HttpPost("/login")]
public async Task<IActionResult> Login([FromBody] LoginModel login)
{
    // Validate user credentials
    if (isValidLogin(login))
    {
        var claims = new Claim[]
        {
            new Claim(JwtRegisteredClaimNames.Sub, login.Username),
            new Claim("UserId", "12345") // Replace with actual user ID
        };

        var token = GenerateJwtToken(claims);
        return Ok(new { token });
    }
    else
    {
        return Unauthorized();
    }
}
```

This code defines a logging API endpoint. It gets user credentials in the request body and validates them first. If it is genuine, it generates a JSON Web Token (JWT) and returns it.

Secure API Endpoints

To API endpoints that require access control, use the "Authenticated" authorization policy.

Example of Secure API Endpoints

```
[HttpGet("/protected-data")]
[Authorize(Policy = "Authenticated")]
public IActionResult GetProtectedData()
{
    // Access and return protected data based on user claims
}
```

This code defines an API endpoint that is only accessible to authorized users. It initially looks for a valid JWT token with the "Authenticated" policy by checking the "Authorization" header.

Using npm, install required packages such as @angular/common/http and jwt-decode.

Create Auth Service

Create a service that handles user authentication, token storage, and request authorization. Send login requests with HttpClient and save the obtained token to localStorage.

Example of Create Auth Service

```
@Injectable({ providedIn: 'root' })

export class AuthService {
  constructor(private http: HttpClient) {}

  login(username: string, password: string): Observable<any> {
    return this.http.post('/login', { username, password });
  }

  setToken(token: string) {
    localStorage.setItem('token', token);
  }

  getToken() {
    return localStorage.getItem('token');
  }

  isAuthenticated() {
    const token = this.getToken();
    return !!token && jwtDecode(token).exp > Date.now() / 1000;
  }

  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpResponse<any>> {
    const token = this.getToken();
    if (token) {
      req = req.clone({ setHeaders: { Authorization: `Bearer ${token}` } });
    }
    return next.handle(req);
  }
}
```

This code defines a service used in an application to manage user authentication. It injects the HttpClient service, which is used to make API calls. It includes methods for logging in, setting a token, getting a token, and checking if the user is authenticated. It also includes an interceptor to add the token to each request header.

Use Auth Service in Components

Example of using Auth Service in Components

```
@Component({
  selector: 'app-protected-data',
  templateUrl: './protected-data.component.html',
})
export class ProtectedDataComponent implements OnInit {
  constructor(private authService: AuthService) {}
  ngOnInit(): void {
    if (!this.authService.isAuthenticated()) {
      // Redirect to login page
    } else {
      // Call protected API endpoint and display data
    }
  }
}
```

This code defines the ProtectedDataComponent component, which shows protected data. On initialization, it checks to see if the user is authenticate

Generate JWT

- The process of establishing a JSON Web Token (JWT) on the server side in ASP.NET Core 2.1, which is used for token-based authentication in Angular.
- This token is used to securely store and transfer user identity information between the client and server without the need to send credentials again.

Example of Generating JWT

```
[Route("api/login")]
public class LoginController : Controller
{
  [HttpPost]
  public IActionResult Login([FromBody] LoginModel model)
  {
    // Validate user credentials
    if (model.Username == "validUser" && model.Password == "validPassword")
    {
      // Create claims for user roles
      var claims = new[]
      {
        new Claim(ClaimTypes.Name, model.Username),
        new Claim(ClaimTypes.Role, "Admin")
      };

      // Create JWT token using claims and secret key
      var token = new JwtSecurityToken(
        issuer: "YourIssuer",
        audience: "YourAudience",
        claims: claims,
        expires: DateTime.UtcNow.AddMinutes(30), // Set expiry time
        signingCredentials: new SigningCredentials(
          new SymmetricSecurityKey(Encoding.UTF8.GetBytes("YourSecretString")),
          SecurityAlgorithms.HmacSha256
      );
    }
  }
}
```



Live Batches



Masterclasses

Menu



Free Courses

Account

```

        return Ok(new { token = new JwtSecurityTokenHandler().WriteToken(token) });
    }
    return Unauthorized();
}
}

```

This code defines a web API login endpoint. When it receives credentials, it compares them to a pre-defined "validUser" and password combination. If

Client App in Angular 6

The "Client App" refers to the single-page application (SPA) produced with Angular 6 that interacts with the API server to access protected resources.

Example of Client App in Angular 6

Startup.cs

```

public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    ...
    ...
    app.UseMvc(routes =>
    {
        routes.MapRoute(
            name: "default",
            template: "{controller}/{action=Index}/{id?}");
    });

    app.UseSpa(spa =>
    {
        // To learn more about options for serving an Angular SPA from ASP.NET Core,
        // see https://go.microsoft.com/fwlink/?linkid=864501

        spa.Options.SourcePath = "ClientApp";

        if (env.IsDevelopment())
        {
            spa.UseAngularCliServer(npmScript: "start");
        }
    });
}

```

This code sets up an ASP.NET Core project to provide server-side MVC routes as well as a client-side Angular SPA. It creates a default MVC route before

```

Startup.cs package.json X
Schema: http://json.schemastore.org/package
1  {
2      "name": "tokenbasedauthentication",
3      "version": "0.0.0",
4      "license": "MIT",
5      "scripts": {
6          "ng": "ng",
7          "start": "ng serve --extract-css",
8          "build": "ng build --extract-css",
9          "build:ssr": "npm run build -- --app=ssr --output-hashing=media",
10         "test": "ng test",
11         "lint": "ng lint"
12     }
13 }

```

Live Batches

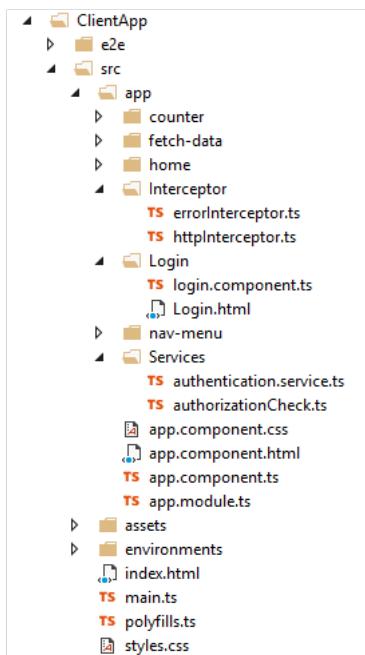
Masterclasses

Menu

Free Courses

Account

I have created the following code structure for the Angular application. Here, I have created all client code under the "scr\app" folder. I have created a lot of components and services.



Authentication Service

- The Authentication Service manages user login, token creation, and API interaction in token-based authentication using Angular 6 and ASP.NET Core 2.1.
- It serves as a link between the user interface & the backend API, managing authentication and securing access to protected resources.

Example of Authentication Service

```

import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { map } from 'rxjs/operators';

@Injectable()
export class AuthenticationService {
  constructor(private http: HttpClient) { }

  login(username: string, password: string) {
    return this.http.post<any>('/api/login', { username, password })
      .pipe(map(user => {
        // login successful if there's a jwt token in the response
        if (user && user.token) {
          // store user details and jwt token in local storage to keep user logged in between page refreshes
          localStorage.setItem('TokenInfo', JSON.stringify(user));
        }
      })

    return user;
  });

  logout() {
    // remove user from local storage to log user out
    localStorage.removeItem('TokenInfo');
  }
}

```

This code defines an Angular service for token-based user login and logout. It transmits login credentials to the API, parses the response for a token, and stores it in local storage.

Login Template and component

- The login component template contains username and password fields.
- It also shows the validation for invalid fields when you click the submit button.
- The submit event of the form is bound to the "OnLogin" method of the login component.
- The "OnLogin" method uses the authentication service's "login" method to validate user credentials and generate tokens.

Example of Login Template

```
<h2>Login</h2>
<form [formGroup]="loginForm" (ngSubmit)="onLogin()">
  <div class="form-group">
    <label for="username">Username</label>
    <input type="text" formControlName="username" class="form-control" [ngClass]="{{ 'is-invalid': submitted && formData.username.errors }}"/>
    <div *ngIf="submitted && formData.username.errors" class="invalid-feedback">
      <div *ngIf="formData.username.errors.required">Username is required</div>
    </div>
  </div>
  <div class="form-group">
    <label for="password">Password</label>
    <input type="password" formControlName="password" class="form-control" [ngClass]="{{ 'is-invalid': submitted && formData.password.errors }}"/>
    <div *ngIf="submitted && formData.password.errors" class="invalid-feedback">
      <div *ngIf="formData.password.errors.required">Password is required</div>
    </div>
  </div>
  <div class="form-group">
    <button [disabled]="submitClick" class="btn btn-primary">Login</button>
  </div>
  <div *ngIf="error" class="alert alert-danger">{{error}}</div>
</form>
```

This code example shows a login form with fields for username and password. When submitted, it validates inputs and displays error messages for required fields.

Example of Login Component

```
import { Component, OnInit } from '@angular/core';
import { Router, ActivatedRoute } from '@angular/router';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { first } from 'rxjs/operators';

import { AuthenticationService } from '../Services/authentication.service';

@Component({
  templateUrl: 'login.html'
})
export class LoginComponent implements OnInit {

  loginForm: FormGroup;
  submitClick = false;
```

```

error = '';

constructor(
  private formBuilder: FormBuilder,
  private route: ActivatedRoute,
  private router: Router,
  private authenticationService: AuthenticationService) { }

ngOnInit() {
  this.loginForm = this.formBuilder.group({
    username: ['', Validators.required],
    password: ['', Validators.required]
  });

  // reset login status
  this.authenticationService.logout();

  // get return url from route parameters or default to '/'
  this.returnUrl = this.route.snapshot.queryParams['returnUrl'] || '/';
}

// convenience getter for easy access to form fields
get formData() { return this.loginForm.controls; }

onLogin() {
  this.submitted = true;

  // stop here if form is invalid
  if (this.loginForm.invalid) {
    return;
  }

  this.submitClick = true;
  this.authenticationService.login(this.formData.username.value, this.formData.password.value)
    .pipe(first())
    .subscribe(
      data => {
        this.router.navigate([this.returnUrl]);
      },
      error => {
        this.error = error;
        this.submitClick = false;
      });
}
}

```

This Angular component defines a login form, performs login logic, and redirects based on success or failure. It uses reactive forms, tracks form subr

Restrict Unauthenticated Users to Access the Application

- Securing your app from unauthorized access in Angular 6 and ASP.NET Core 2.1 token-based authentication requires an approach that involves tv



Live Batches



Masterclasses



Free Courses

Account

- This ensures that only authenticated users with a valid token can access protected sections, preventing unauthorized people from entering.

Example of Restrict Unauthenticated Users to Access the Application

```
import { Injectable } from '@angular/core';
import { Router, CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot } from '@angular/router';

@Injectable()
export class AuthorizationCheck implements CanActivate {

  constructor(private router: Router) { }

  canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot) {
    // If token data exist, user may login to application
    if (localStorage.getItem('TokenInfo')) {
      return true;
    }

    // otherwise redirect to login page with the return url
    this.router.navigate(['/login'], { queryParams: { returnUrl: state.url } });
    return false;
  }
}
```

This code defines an Angular service called "AuthorizationCheck" that searches localStorage for a user's token. If it exists, they are authorized to use the application. Otherwise, they are redirected to the login page with the return URL.

HTTP Interceptor

- Token-based authentication in Angular 6 and ASP.NET Core 2.1 relies on Http Interceptors to securely pass the user's JWT token with each request.
- Consider a secret agent (Interceptor) intercepting all outgoing communications (requests), attaching the hidden access key (token) to each, and the server recognizes the key and grants access, and the agent responds.
- This guarantees that only authorized users have access to protected resources, keeping your data safe.

Example of HTTP Interceptor

```
import { Injectable } from '@angular/core';
import { HttpRequest, HttpHandler, HttpEvent, HttpInterceptor } from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable()
export class httpInterceptor implements HttpInterceptor {

  intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    // add authorization header to request

    //Get Token data from local storage
    let tokenInfo = JSON.parse(localStorage.getItem('TokenInfo'));

    if (tokenInfo && tokenInfo.token) {
      request = request.clone({
        setHeaders: {
          Authorization: `Bearer ${tokenInfo.token}`,
          'Content-Type': 'application/x-www-form-urlencoded; charset=utf-8'
        }
      });
    }

    return next.handle(request);
  }
}
```

```

    }

    return newRequest.handle(request);
}

}

```

If a valid token exists, this code inserts an authorization header with the user's token from localStorage to every HTTP request. It simply injects the "Bearer" prefix and the token value.

Error Interceptor

- Error Interceptor serves as a consistent guard in Angular 6 and ASP.NET Core 2.1 token-based authentication.
- It intercepts all incoming answers and looks for problems in unauthorized access (401).
- If it detects unauthorized behavior, it immediately revokes the user's token and redirects them to the login page, protecting your app from unauthorized access.

Example of Error Interceptor

```

import { Injectable } from '@angular/core';
import { HttpRequest, HttpHandler, HttpEvent, HttpInterceptor } from '@angular/common/http';
import { Observable } from 'rxjs/Rx';
import { catchError } from 'rxjs/operators';

import { AuthenticationService } from '../Services/authentication.service';
import { Router } from '@angular/router';

@Injectable()
export class ErrorInterceptor implements HttpInterceptor {
  constructor(private authenticationService: AuthenticationService, private router: Router) { }

  intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    return newRequest.handle(request).pipe(catchError(err =>{
      if (err.status === 401) {
        //if 401 response returned from api, logout from application & redirect to login page.
        this.authenticationService.logout();
      }

      const error = err.error.message || err.statusText;
      return Observable.throw(error);
    }));
  }
}

```

This code intercepts HTTP errors, notably 401 (unauthorized access). If it is discovered, it initiates an automatic logout via the AuthenticationService and then redirects the user to the login page.

App Module and Routing

- The App Module serves as the control center for token-based authentication in Angular 6 & ASP.NET Core 2.1.
- It registers all authentication components, services, and interceptors, including the routing logic.
- The paths for protected and public areas are defined via routing, which ensures that only authorized users have access to sensitive data.

Example of App Module and Routing



Live Batches



Masterclasses



Free Courses

Account

```

import { FormsModule, ReactiveFormsModule } from '@angular/forms';
import { HttpClientModule, HTTP_INTERCEPTORS } from '@angular/common/http';
import { RouterModule } from '@angular/router';

import { AppComponent } from './app.component';
import { NavMenuComponent } from './nav-menu/nav-menu.component';
import { HomeComponent } from './home/home.component';
import { CounterComponent } from './counter/counter.component';
import { FetchDataComponent } from './fetch-data/fetch-data.component';

import { LoginComponent } from './Login/login.component'

import { httpInterceptor } from './Interceptor/httpInterceptor';
import { ErrorInterceptor } from './Interceptor/errorInterceptor';

import { AuthorizationCheck } from './Services/authorizationCheck'
import { AuthenticationService } from './Services/authentication.service'

@NgModule({
  declarations: [
    AppComponent,
    NavMenuComponent,
    HomeComponent,
    CounterComponent,
    FetchDataComponent,
    LoginComponent
  ],
  imports: [
    BrowserModule.withServerTransition({ appId: 'ng-cli-universal' }),
    HttpClientModule,
    FormsModule,
    ReactiveFormsModule,
    RouterModule.forRoot([
      { path: '', component: HomeComponent, pathMatch: 'full', canActivate: [AuthorizationCheck] },
      { path: 'counter', component: CounterComponent, canActivate: [AuthorizationCheck] },
      { path: 'fetch-data', component: FetchDataComponent, canActivate: [AuthorizationCheck] },
      { path: 'login', component: LoginComponent }
    ])
  ],
  providers: [
    { provide: HTTP_INTERCEPTORS, useClass: httpInterceptor, multi: true },
    { provide: HTTP_INTERCEPTORS, useClass: ErrorInterceptor, multi: true },
    AuthorizationCheck,
    AuthenticationService],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

This code creates the authentication core for your Angular project. It establishes routes, and components (login, navigation), and protects regions. It also includes interceptors for handling HTTP requests.

Read More:

- [Angular Developer Salary In India](#)
- [Angular Developer Skills](#)



Live Batches



Masterclasses



Free Courses

Account

Summary

Token-based authentication is a safe and efficient method of managing user access in current web applications. You may ensure a safe and seamless

Do you Know?

.NET is gaining popularity day by day, especially after the release of .NET 8. .NET 8 is not only a framework version but much more than that. It redefines

Therefore, if you want to upskill yourselves and stand out from others consider our following training programs on .NET.

- [.NET Developer Training With Certification](#)
- [ASP.NET Core Certification Training](#)
- [ASP.NET Core Course](#)
- [.NET Solution Architect Certification Training](#)
- [Full-Stack .NET Developer Certification Training Program](#)
- [Advanced Full-Stack .NET Developer Certification Training](#)

FAQs

Q1. How does token-based authentication work in Net core?

Add the following to your ConfigureServices function to configure ASP.NET Core to utilize token authentication as the default authentication strategy

Q2. How does token-based authentication work in Angular?

Q3. What's the distinction between OAuth 2.0 & JWT?

Q4. In Angular, what is JWT token authentication?

Q5. What is token-based authentication?

In less than 5 minutes, with

[GET FREE CHALLENGE](#)



Similar Articles

[Tips to Secure Your Angular Application](#)

[Is Angular Certification Worth It in 2025? Top 10 Certifications Compared](#)

[Angular vs. React vs Blazor: A Detail Comparison 2025](#)

[How Long Does it Take to Learn Angular?](#)

[Angular Roadmap to become an Expert](#)

[Angular vs. React vs. Vue: Which is Best?](#)



Live Batches



Masterclasses

Menu



Free Courses

Account

[◀ Previous Tutorial](#)

Angular 19 Tutorial: Complete Roadmap to Learn Angular 19

[Next Tutorial ▶](#)

Top 60+ Angular Interview Question & Answers

About Author



Jignesh Trivedi (Microsoft MVP and Technical Lead)

[View Profile](#)

Jignesh Trivedi is working as a software developer with a leading organization and having more than 11 years of experience. He is very passionate about Microsoft Technologies. He is author, speaker and MVP.

He has the experience to develop enterprise application using Microsoft technologies such as ASP.NET Core, C#, SQL Server, etc. and other technologies such as JavaScript, Angular, Node.js, etc. He loves building great products and POC (proof of concepts) using the best available technologies. He loves to share his knowledge by contributing to the Developer community.

Our Courses

Angular Foundations Course Free

Free Angular Online Course for Beginners! Master Angular, create real-world projects, and earn your certificate. Start learning in 2025—register now!

[View More](#)

Node.js Foundations Course Free

Node.js Free Online Course with Certification in 2025! Perfect for beginners. Master backend skills, get certified, and boost your career now. Enroll today

[View More](#)

TypeScript Programming Course For Beginners

Free

Free TypeScript Online Course with Certification in 2025! Designed for beginners, learn step-by-step and get certified. Sign up now to get started!

[View More](#)

Angular Certification Training

Enroll in the best Angular certification course online. Learn with live projects, hands-on labs, and 100% job assistance for career growth. ✓ Unlimited Live Sessions ✓ Project-based Training ✓ Job Assistance ✓ Enrol Now!

[View More](#)


A world-class EdTech platform



Masterclasses

Unlimited live sessions

600+ Quick notes



Live Batches



Free Courses

Menu

Account

Job Oriented Training

Azure AI & Gen AI Engineer Training | AWS AI & Gen AI Engineer Training | Advanced Full-Stack Java Developer Training | Full-Stack .NET Developer Training | Advanced Full-Stack .NET Training | .NET Solution Architect Training | .NET & Azure Solution Architect Training | Java Solution Architect Training | Java Microservices with AWS Training

Popular Live Training

Azure AI Engineer Training | Azure GenAI/Agentic AI Training | ASP.NET Core Training | .NET Software Architecture & Design Training | Java Software Architecture Training | Angular Training | React Training | Azure Developer Training | AWS Developer Training | .NET Microservices Training | Java Microservices Training

Free Courses

Azure Free Course | DSA Free Course | Java Free Course | Python Free Course | C++ Free Course | C# Free Course | C Free Course | JavaScript Free Course | TypeScript Free Course | SQL Server Free Course | Html Free Course | Node.js Free Course | MongoDB Free Course | ASP.NET MVC Free Course | All Free Courses

Free Interview Books

ASP.NET Core Interview Book | Azure Interview Book | Angular Interview Book | React Interview Book | .NET Microservices Interview Book | DSA Interview Book | Java Interview Book | Python Interview Book | C# Interview Book | C++ Interview Book | Design Patterns Interview Book | All Free Books

Popular Tutorial

Azure Cloud Tutorial | .NET Tutorial | ASP.NET Core Tutorial | Angular Tutorial | React Tutorial | Microservices Tutorial | Design Patterns Tutorial | DevOps Tutorial | SQL Server Tutorial | DSA Tutorial | Python Tutorial | C++ Tutorial | Java Tutorial | C# Tutorial | JavaScript Tutorial | Job Interview Articles | AI Tutorial

Platform

Live Training
Free Courses
Hands-on Labs
Real-World Projects
DSA Problems
Quick Notes
Free Interview Books
Corporate Training

Company

About Us
Contact Us
Terms & Conditions
Privacy Policy
Refund Policy
Subscription Policy
Verify Certificate
Become An Instructor

Resources

Live Training Membership
Master Classes
Coding Playground
Skill Tests
Job Openings
Mentors

Have any Questions?

+91- 999 9123 502
hello@scholarhat.com
Follow Us



Live Batches



Masterclasses



Free Courses

Account

© 2025 Dot Net Tricks Innovation Pvt. Ltd. | All rights reserved | The course names and logos are the trademarks of their respective owners | Engineered with in India.