

The screenshot shows the VS Code interface with the title bar "Angular Login & Logout with Asp.Net Core Identity & JWT" and the status bar indicating "AuthECCClient". The Explorer sidebar on the left lists files and folders: .angular, .vscode, node_modules, public, src, app, shared, user, app.component.html, app.component.ts, app.config.ts, app.routes.ts, index.html, main.ts, styles.css, .editorconfig, .gitignore, angular.json, app-structure.txt (selected), package-lock.json, package.json, README.md, and tsconfig.app.json. The center pane displays the file "app-structure.txt" which contains the following content:

```
app
└── app
    ├── user
    │   ├── user.component.ts|html
    │   ├── login (cmp. folder)      **
    │   └── registration (cmp. folder)
    ├── dashboard (cmp. folder)  **
    └── shared
        ├── pipes
        │   └── first-key.pipe.ts
        └── services
            └── auth.service.ts
```

The bottom status bar shows "2:10 / 1:42:40 • Introduction & Login Form Design >" and various editor settings like "Ln 8, Col 36", "Spaces: 2", "UTF-8", "CRLF", "Plain Text", and "Go To". A YouTube video player is visible at the bottom right.

EXPLORER: AUTHECLIENT

```
└── app
    ├── user
    │   ├── user.component.ts|.html
    │   ├── login (cmp. folder)      **
    │   └── registration (cmp. folder)
    ├── dashboard (cmp. folder)   **
    └── shared
        └── nines
```

PROBLEMS PORTS OUTPUT DEBUG CONSOLE TERMINAL

```
CodAffection@CodAffection MINGW64 /d/YouTube/Videos/Angular/Angular 18/Login wi
th .Net Core API Angular 18 (2024)/Project/AuthECClient
$ ng g c user/login
```

tsconfig.app.json

Create dashboard

Angular Login & Logout with Asp.Net Core Identity & JWT

```
└── app
    ├── user
    │   ├── user.component.ts|.html
    │   ├── login (cmp. folder)      **
    │   └── registration (cmp. folder)
    ├── dashboard (cmp. folder)   **
    └── shared
        └── nines
```

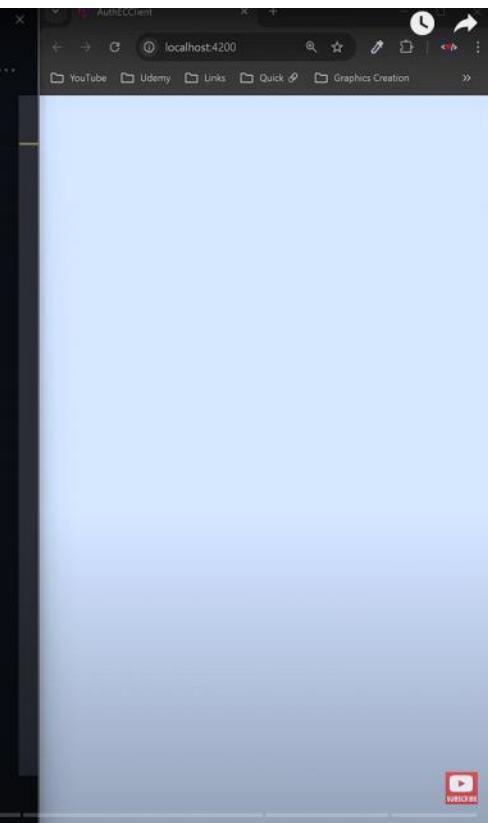
PROBLEMS PORTS OUTPUT DEBUG CONSOLE TERMINAL

```
CREATE src/app/user/login/login.component.html (21 bytes)
CREATE src/app/user/login/login.component.ts (219 bytes)

CodAffection@CodAffection MINGW64 /d/YouTube/Videos/Angular/Angular 18/Login wi
th .Net Core API Angular 18 (2024)/Project/AuthECClient
$ ng g c dashboard
CREATE src/app/dashboard/dashboard.component.html (25 bytes)
CREATE src/app/dashboard/dashboard.component.ts (231 bytes)

CodAffection@CodAffection MINGW64 /d/YouTube/Videos/Angular/Angular 18/Login wi
th .Net Core API Angular 18 (2024)/Project/AuthECClient
```

For lots of routes

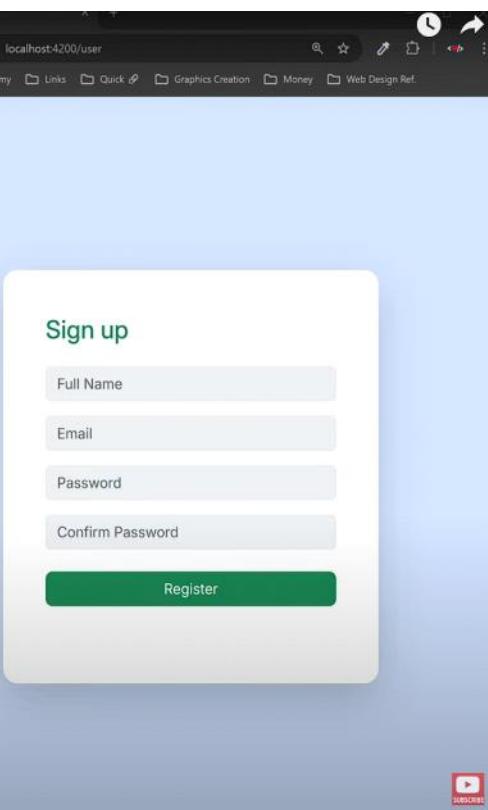


Angular Login & Logout with Asp.Net Core Identity & JWT

app-structure.txt app.component.html app.component.ts

Go to component

```
<router-outlet>
```



Angular Login & Logout with Asp.Net Core Identity & JWT

app-structure.txt app.component.html app.routes.ts app.component.ts

```
import { Routes } from '@angular/router';
import { UserComponent } from './user/user.component';

export const routes: Routes = [
  {path: 'user', component: UserComponent}
];
```

Angular Login & Logout with Asp.Net Core Identity & JWTClient

The screenshot shows a code editor with multiple tabs: app.component.html, app.routes.ts, user.component.html (active), and app.component.ts. The user.component.html tab contains the following code:

```

<div class="container">
  <div class="row vh-100 align-items-center">
    <div class="col-lg-4 offset-lg-4 col-md-8 offset-md-2 col-10 offset-1">
      <div class="rounded-4 border-0 shadow-lg bg-white" style="min-height:450px;">
        <div class="d-flex justify-content-start align-items-center">
          <div class="p-5 w-100">
            <router-outlet />
          </div>
        </div>
      </div>
    </div>
  </div>
</div>

```

A browser window on the right displays a registration form with fields for Email, Password, and Confirm Password, and a Register button. A tooltip message from Angular is visible in the browser's UI, stating: "NG8001: 'router-outlet' is not a known element: 1. If 'router-outlet' is an Angular component, then verify that it is included in the '@Component.imports' of this component. 2. If 'router-outlet' is a Web Component then add 'CUSTOM_ELEMENTS_SCHEMA' to the '@Component.schemas' of this component to suppress this message." The URL in the browser is localhost:4200/user.

Import router outlet

Angular Login & Logout with Asp.Net Core Identity & JWTClient

The screenshot shows a code editor with multiple tabs: user.component.html, app.routes.ts, user.component.html (active), and user.component.ts. The user.component.ts tab contains the following code:

```

import { Component } from '@angular/core';
import { RegistrationComponent } from './registration/registration.component';
import { RouterOutlet } from '@angular/router';

@Component({
  selector: 'app-user',
  standalone: true,
  imports: [RegistrationComponent, RouterOutlet],
  templateUrl: './user.component.html',
  styles: []
})
export class UserComponent {
}

```

A browser window on the right shows a blank white page. The URL in the browser is localhost:4200/user.

The screenshot shows a code editor and a browser side-by-side. The code editor on the left displays `app.routes.ts` with the following content:

```
import { Routes } from '@angular/router';
import { UserComponent } from './user/user.component';
import { RegistrationComponent } from './user/registration/registration.component';
import { LoginComponent } from './user/login/login.component';

export const routes: Routes = [
  {path:'user',component:UserComponent,
  children:[
    {path:'signup',component:RegistrationComponent},
    {path:'login',component:LoginComponent},
  ]
}
];
```

The browser window on the right shows a simple page with the URL `localhost:4200/user/login`. A single line of text, "login works!", is displayed in the center of the page.

Angular Login & Logout with Asp.Net Core Identity & JWT

```
app.routes.ts X login.component.html user.component.html user.component.t ...
```

```
import { Routes } from '@angular/router';
import { UserComponent } from './user/user.component';
import { RegistrationComponent } from './user/registration/registration.component';
import { LoginComponent } from './user/login/login.component';

export const routes: Routes = [
  {path: '', component: UserComponent,
    children: [
      {path: 'signup', component: RegistrationComponent},
      {path: 'login', component: LoginComponent},
    ]
  }
];
```

AuthClient

localhost:4200/login

login works!



6:21 / 1:42:40 - Introduction & Login Form Design

Angular Login & Logout with Asp.Net Core Identity & JWT

```
app.routes.ts login.component.ts login.component.html user.component.html ...
```

```
import { CommonModule } from '@angular/common';
import { Component } from '@angular/core';
import { ReactiveFormsModuleModule } from '@angular/forms';

@Component({
  selector: 'app-login',
  standalone: true,
  imports: [CommonModule, ReactiveFormsModuleModule],
  templateUrl: './login.component.html',
  styles: ``
})
export class LoginComponent {
```

AuthClient

localhost:4200/login

login works!



6:21 / 1:42:40 - Introduction & Login Form Design

The screenshot shows a development environment with two main panes. The left pane is a code editor for an Angular project, displaying the file `login.component.ts`. The code defines a component named `app-login` with a template URL pointing to `./login.component.html`. It includes a constructor dependency on `FormBuilder` and creates a form group with email and password fields, both requiring validation. The right pane is a browser window titled "AuthClient" showing the URL `localhost:4200/login`. The page displays a simple message: "login works!".

```
Angular Login & Logout with Asp.Net Core Identity & JWT
```

```
login.component.ts X registration.component.ts login.component.html user.css ...
```

```
@Component({
  selector: 'app-login',
  standalone: true,
  imports: [CommonModule, ReactiveFormsModule],
  templateUrl: './login.component.html',
  styles: ``
})

export class LoginComponent {
  constructor(public formBuilder: FormBuilder) { }

  form = this.formBuilder.group({
    email: ['', Validators.required],
    password: ['', Validators.required],
  })
}
```

```
localhost:4200/login
```

```
login works!
```

Angular Login & Logout with Asp.Net Core Identity & JWT

```

Go to component
<div class="mb-4">
  <h2 class="text-success">
    Sign up
  </h2>
  <div>Already have an account?
    <a [routerLink]="/login" class="text-decoration-none fw-medium text-success d-inlin-block">Sign in</a>
  </div>
</div>
<form [formGroup]="form" (ngSubmit)="onSubmit()">
  <div class="mb-3">
    <input class="form-control bg-body-secondary" placeholder="Full Name" formControlName="fullName">
    <div class="error-feedback" *ngIf="hasDisplayableError('fullName') && form.controls.fullName.hasError('required')">
      Please enter your full name.
    </div>
  </div>
  <div class="mb-3">
    <input class="form-control bg-body-secondary" placeholder="Email" formControlName="email">
    <div class="error-feedback" *ngIf="hasDisplayableError('email')">
      Please enter a valid email address.
    </div>
  </div>
  <div class="mb-3">
    <input class="form-control bg-body-secondary" placeholder="Password" type="password" formControlName="password">
    <div class="error-feedback" *ngIf="hasDisplayableError('password')">
      Password must be at least 6 characters long.
    </div>
  </div>
  <div class="mb-3">
    <input class="form-control bg-body-secondary" placeholder="Confirm Password" type="password" formControlName="confirmPassword">
    <div class="error-feedback" *ngIf="hasDisplayableError('confirmPassword')">
      Confirm password does not match.
    </div>
  </div>
  <button type="submit" class="btn btn-primary w-100">Register</button>
</form>

```

Import router link

```

import { CommonModule } from '@angular/common';
import { Component } from '@angular/core';
import { FormBuilder, ReactiveFormsModule, Validators } from '@angular/forms';
import { RouterLink } from '@angular/router';

@Component({
  selector: 'app-login',
  standalone: true,
  imports: [CommonModule, ReactiveFormsModule, RouterLink],
  templateUrl: './login.component.html',
  styles: []
})
export class LoginComponent {
  constructor(public formBuilder: FormBuilder) { }

  form = this.formBuilder.group({
    email: ['', Validators.required],
    password: ['', Validators.required],
  })
}

```

Angular Login & Logout with Asp.Net Core Identity & JWTClient

```
login.component.ts X registration.component.ts login.component.html 1 registration ...
```

```
selector: 'app-login',
standalone: true,
imports: [CommonModule, ReactiveFormsModule, RouterLink],
templateUrl: './login.component.html',
styles: ``
```

```
}
```

```
export class LoginComponent {
constructor(public formBuilder: FormBuilder) { }
isSubmitted: boolean = false;
```

```
form = this.formBuilder.group({
  email: ['', Validators.required],
  password: ['', Validators.required],
})
```

```
hasDisplayableError(controlName: string): Boolean {
  const control = this.form.get(controlName);
  return Boolean(control?.invalid) &&
    (this.isSubmitted || Boolean(control?.touched) || Boolean(control?.dirty))
}
```

Sign in

Don't have an account? [Sign up](#)

Email

Password

Sign in

Angular Login & Logout with Asp.Net Core Identity & JWT Client

The screenshot shows a code editor with several tabs open, including `login.component.ts`, `login.component.html`, and `registration.component.html`. The `login.component.html` tab is active, displaying the following code:

```

<div d-inlin-block>Sign up</a>
</div>
<form [formGroup]="form" (ngSubmit)="onSub()">
  <div class="mb-3">
    <input class="form-control bg-body-secondary" placeholder="Email" formControlName="email">
    <div class="error-feedback" *ngIf="hasDisplayableError('email') && form.controls.email.hasError('required')">
      Please enter your email address.
    </div>
  </div>
  <div class="mb-3">
    <input class="form-control bg-body-secondary" placeholder="Password" formControlName="password">
    <div class="error-feedback" *ngIf="hasDisplayableError('password') && form.controls.password.hasError('required')">
      Please enter your password.
    </div>
  </div>
  <div class="mt-4">
    <button type="submit" class="btn btn-success w-100 rounded-3">
      Sign in
    </button>
  </div>
</form>

```

A browser window is open at `localhost:4200/signin`, showing a "Sign in" form with two input fields ("Email" and "Password") and a "Sign in" button. A red error message at the top states "NG5002: Empty expressions are not allowed" with the URL "src/app/user/login/login.component.html:9:37". A tooltip says "Click outside, press Esc key, or fix the code to dismiss."

Angular Login & Logout with Asp.Net Core Identity & JWT

The screenshot shows a code editor with several tabs open, including `login.component.ts`, `registration.component.ts`, `login.component.html`, `registration.component.html`, `user.component.html`, and `user.co`. The `user.component.html` tab is active, displaying the following code:

```

<div class="container">
  <div class="row vh-100 align-items-center">
    <div class="col-10 offset-1 col-md-8 offset-md-2 col-10 offset-1">
      <d (element) div: HTMLDivElement 0 shadow-lg bg-white" style="min-height:450px;">
        <div class="col-md-5">
          </div>
        <div class="col-md-7 d-flex justify-content-start align-items-center">
          <div class="p-5 w-100">
            <router-outlet />
          </div>
        </div>
      </div>
    </div>
  </div>
</div>

```



Should put this image inside the

Shard/public folder

```
File Edit Selection View Go ... AuthECClient user.component.html x
user.component.ts login.component.html registration.component.html user.component.ts app.component.ts

Go to component
<div class="container">
  <div class="row vh-100 align-items-center">
    <div class="col-lg-4 offset-lg-4 col-md-8 offset-md-2 col-10 offset-1">
      <div class="row rounded-4 border-0 shadow-lg bg-white" style="min-height:450px;">
        <div class="col-md-5 d-md-flex d-none bg-success rounded-start-4 align-items-center justify-content-center">
          <div class="text-center text-white">
            
            <h2 class="fw-bolder">Auth EC</h2>
          </div>
        </div>
        <div class="col-md-7 d-flex justify-content-start align-items-center">
          <div class="p-5 w-100">
            <router-outlet />
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
```

Angular Login & Logout with Asp.Net Core Identity & JWT

AuthECClient

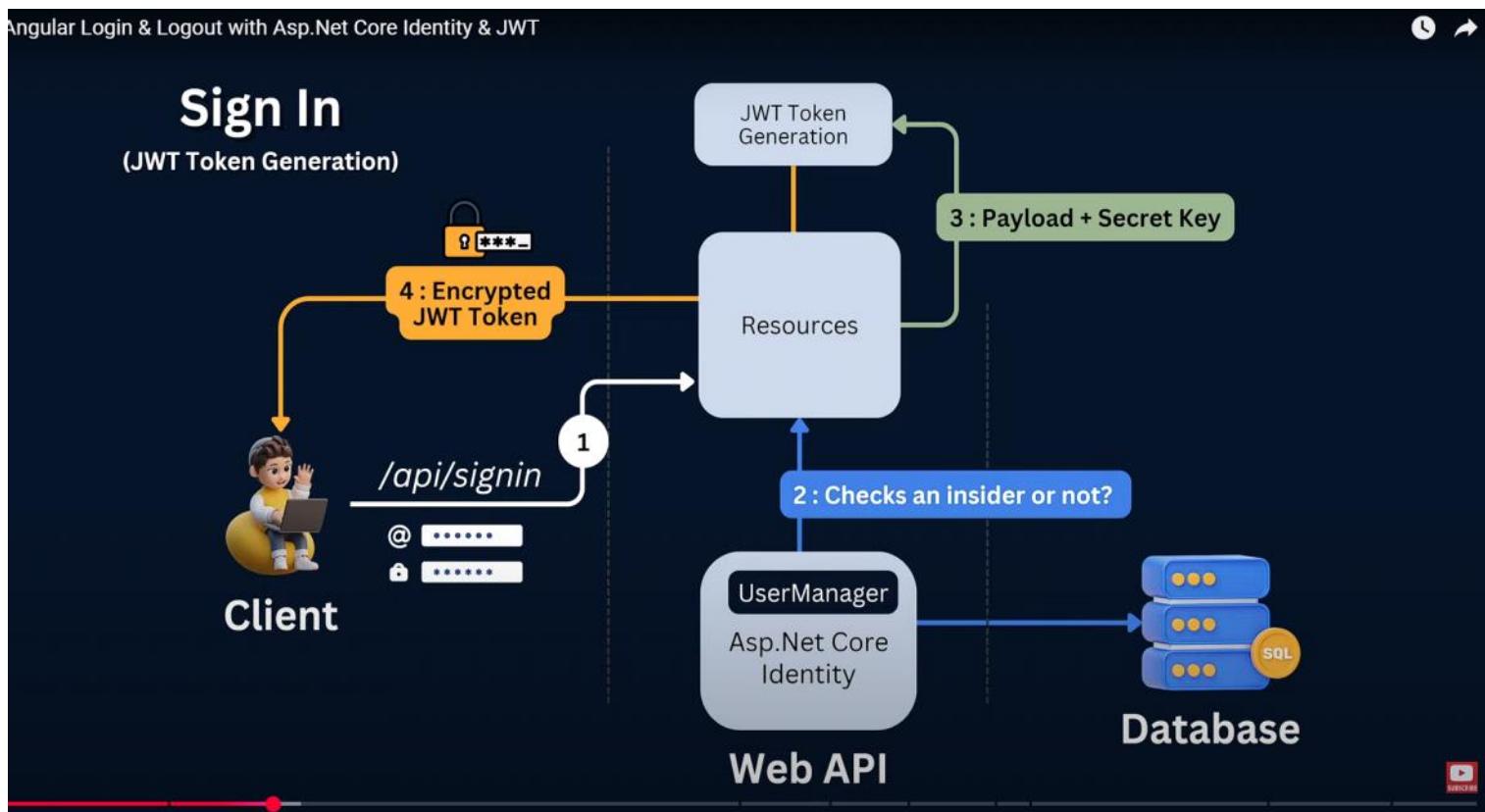
```

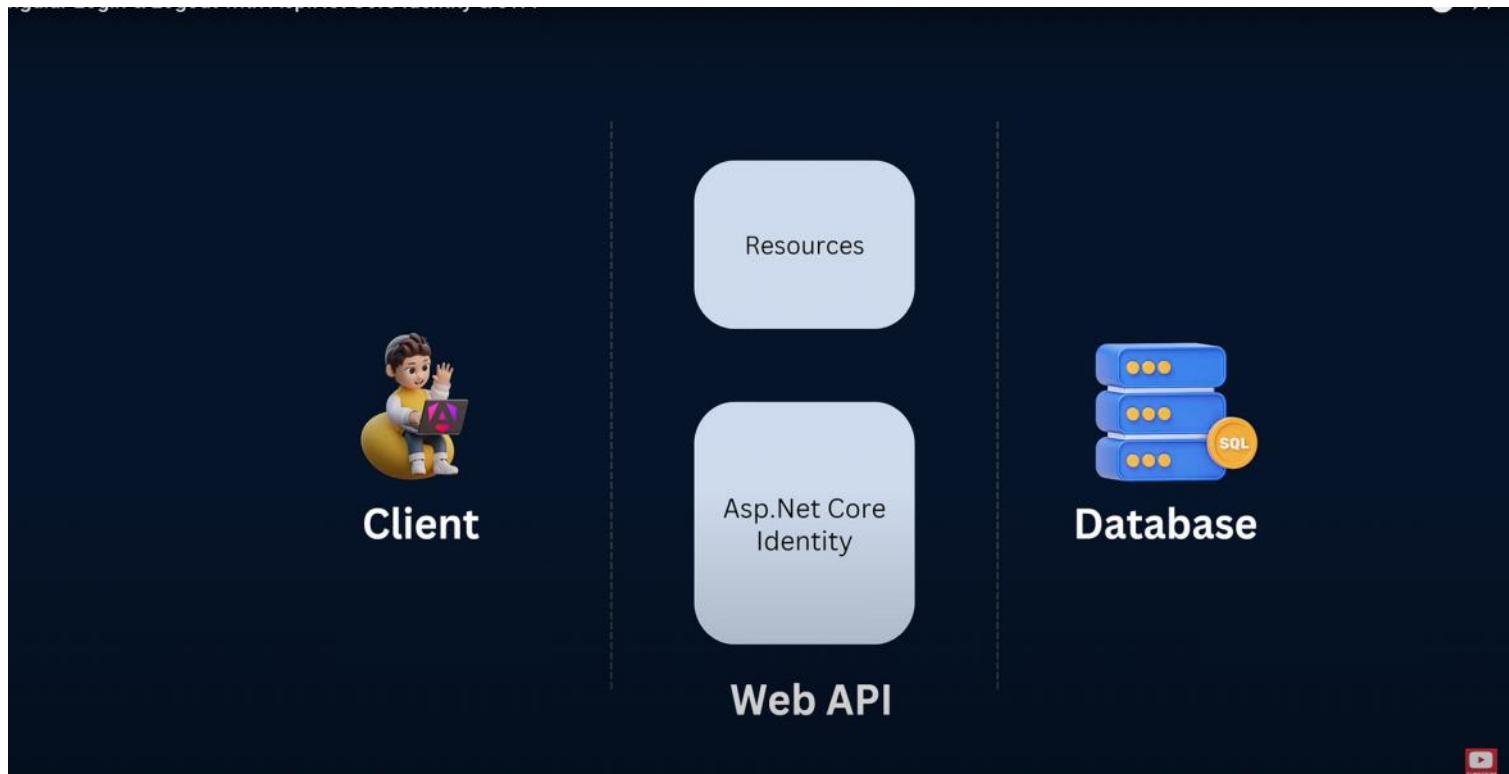
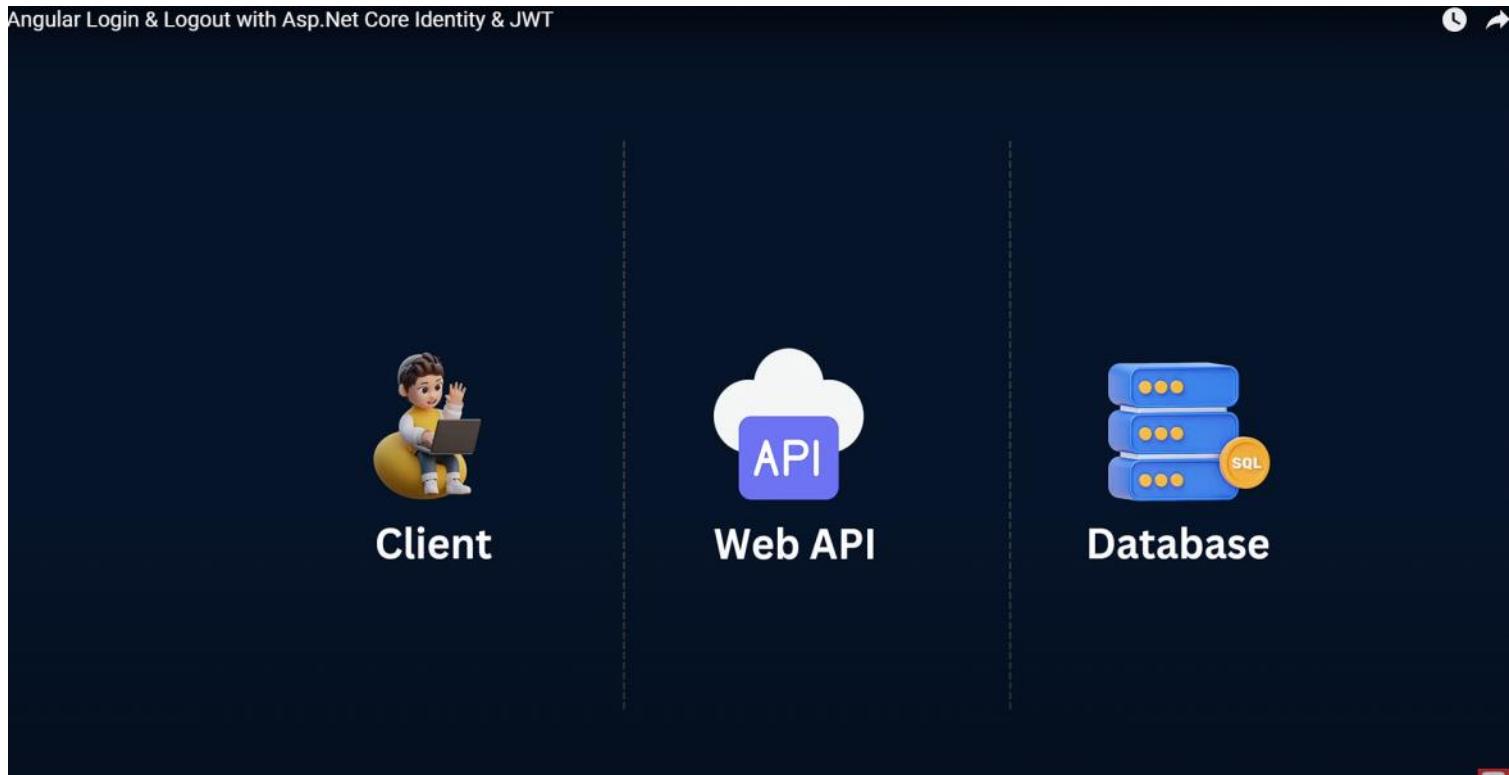
<div class="row vh-100 align-items-center">
  <div class="col-xxl-6 offset-xxl-3 col-lg-8 offset-lg-2 col-10 offset-1">
    <div class="row rounded-4 border-0 shadow-lg bg-white" style="min-height:450px;">
      <div class="col-md-5 d-md-flex d-none bg-success rounded-start-4 align-items-center justify-content-center">
        <div class="text-center text-white">
          
          <h2 class="fw-bolder">Auth EC</h2>
        </div>
      </div>
      <div class="col-md-7 col-12 d-flex justify-content-start align-items-center">
        <div class="p-5 w-100">
          <router-outlet />
        </div>
      </div>
    </div>
  </div>
</div>

```

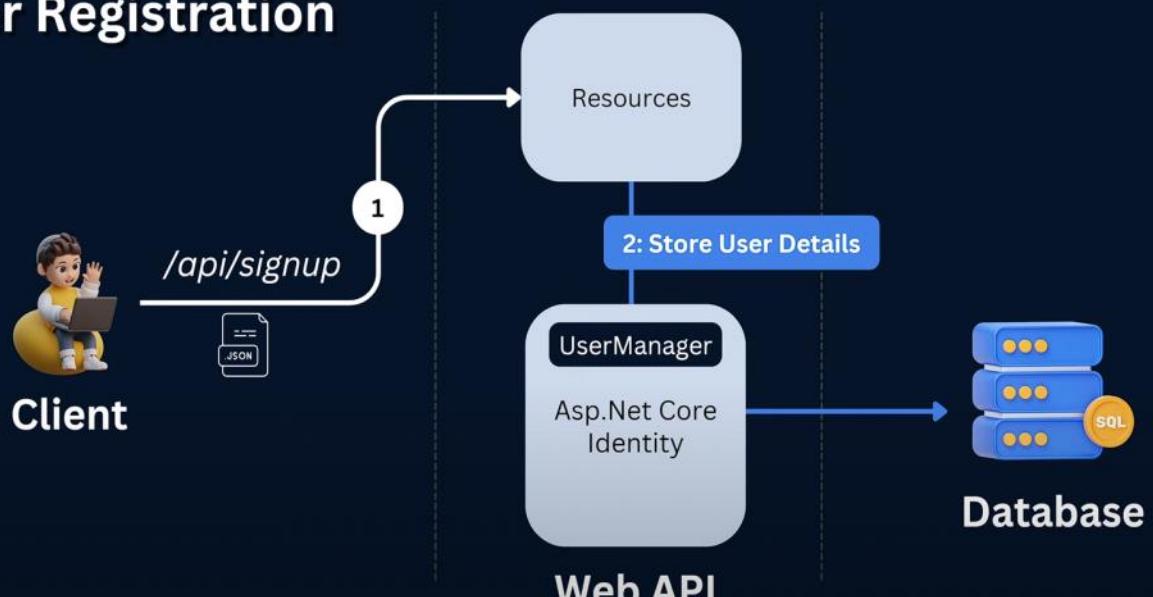
18:12 / 1:42:40 • Login Form Validation >

Ln 3 Col 39 Spaces: 2 UTF-8 CRLF () HTML Go 🔍





User Registration



Angular Login & Logout with Asp.Net Core Identity & JWT

By Mike Rousos

Version

ASP.NET Core in .NET 8.0

Filter by title

- Configure certificate authentication
- Configure Windows authentication
- Configure WS-Federation authentication
- Configure social authentication
- Policy schemes
- Manage JWTs in development
- Map, customize, and transform claims
- Community OSS authentication options
- Identity management solutions
- Multi-factor authentication

Documentation

Generate tokens with `dotnet user-jwts`

Learn how to set up manage JSON Web Tokens in development with `dotnet user-jwts`

Simple authorization in ASP.NET Core

Learn how to use the `Authorize` attribute to restrict access to ASP.NET Core controllers and actions.

Use cookie authentication without ASP.NET Core Identity

Learn how to use cookie authentication without ASP.NET Core Identity.

[Download PDF](#)

Show 3 more

File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help Full Screen

Angular Login & Logout with Asp.Net Core Identity & JWT

Program.cs* AppDbContext.cs WeatherForecast.cs 202407...ity2.cs ApplicationUser.cs appset...gs.json

Solution Explorer

Search Solution Explorer (Ctrl+Shift+F)

Solution 'AuthECAPI' (1 of 1 project)

- Solution Items
- .editorconfig
- AuthECAPI
 - Connected Services
 - Dependencies
 - Properties
 - Controllers
 - Migrations
 - Models
 - AppDbContext.cs
 - ApplicationUser.cs
 - appsettings.json
 - AuthECAPI.http
 - Program.cs
 - WeatherForecast.cs

GitHub Copilot Sign in

```
builder.Services.Configure<IdentityOptions>(options =>
{
    options.Password.RequireDigit = false;
    options.Password.RequireUppercase = false;
    options.Password.RequireLowercase = false;
    options.User.RequireUniqueEmail = true;
});

builder.Services.AddDbContext<AppDbContext>(options =>
    options.UseSqlServer(builder.Configuration.GetConnectionString("Dev"));

builder.Services.AddAuthentication();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}
```

21:30 / 1:42:40 • JWT User Authentication in .Net Core API >

Middlewares

(Asp.Net Core)

**Client**

Asp.Net Core

Angular Login & Logout with Asp.Net Core Identity & JWT

AuthECAPISign inGitHub CopilotGit Changes

```
File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help Search AuthECAPI
Program.cs AppDbC...ext.cs Weather...ller.cs 202407...ity2.cs ApplicationUser.cs appset...gs.json
AuthECAPI
builder.Services.AddDbContext<AppDbContext>(options =>
    options.UseSqlServer(builder.Configuration.GetConnectionString("DevDB"))

builder.Services.AddAuthentication();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

Config.CORS
app.UseAuthentication();
app.UseAuthorization();

app.MapControllers();
```

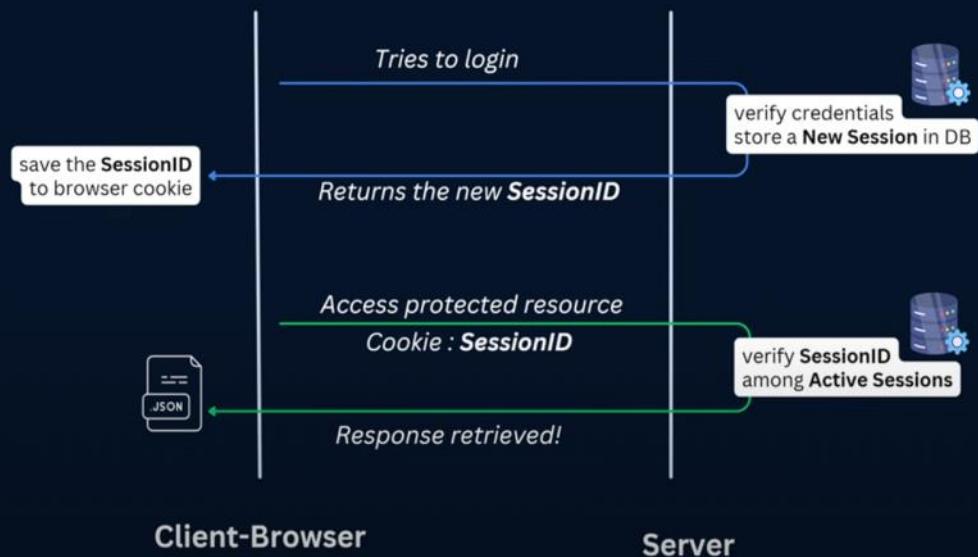
Solution Explorer

- Search Solution Explorer (Ctrl+.)
- Solution 'AuthECAPI' (1 of 1 project)
 - Solution Items
 - .editorconfig
 - AuthECAPI
 - Connected Services
 - Dependencies
 - Properties
 - Controllers
 - Migrations
 - Models
 - AppDbContext.cs
 - AppUser.cs
 - appsettings.json
 - AuthECAPI.http
 - Program.cs
 - WeatherForecast.cs

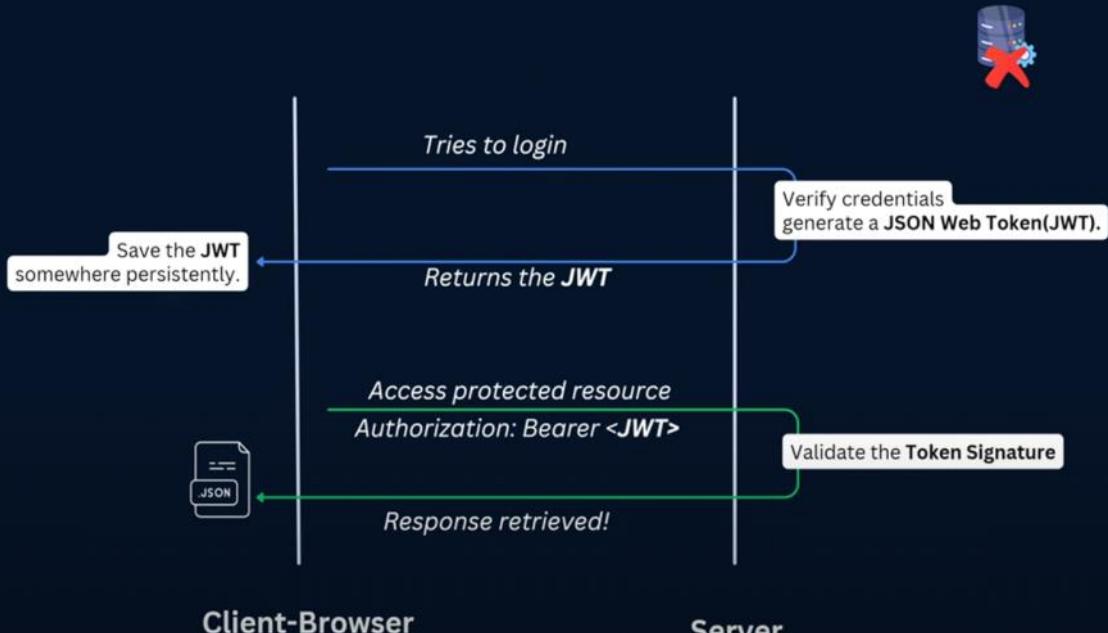
Output

24:32 / 1:42:40 - JWT User Authentication in .Net Core API

Session-Based (Cookie-Based) Authentication



Token-Based Authentication



Structure of JWT



Debugger Libraries Introduction Ask

Crafted by Auth0 by Okta

Header

The header *typically* consists of two parts: the type of the token, which is JWT, and the signing algorithm being used, such as HMAC SHA256 or RSA.

For example:

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

Then, this JSON is **Base64Url** encoded to form the first part of the JWT.

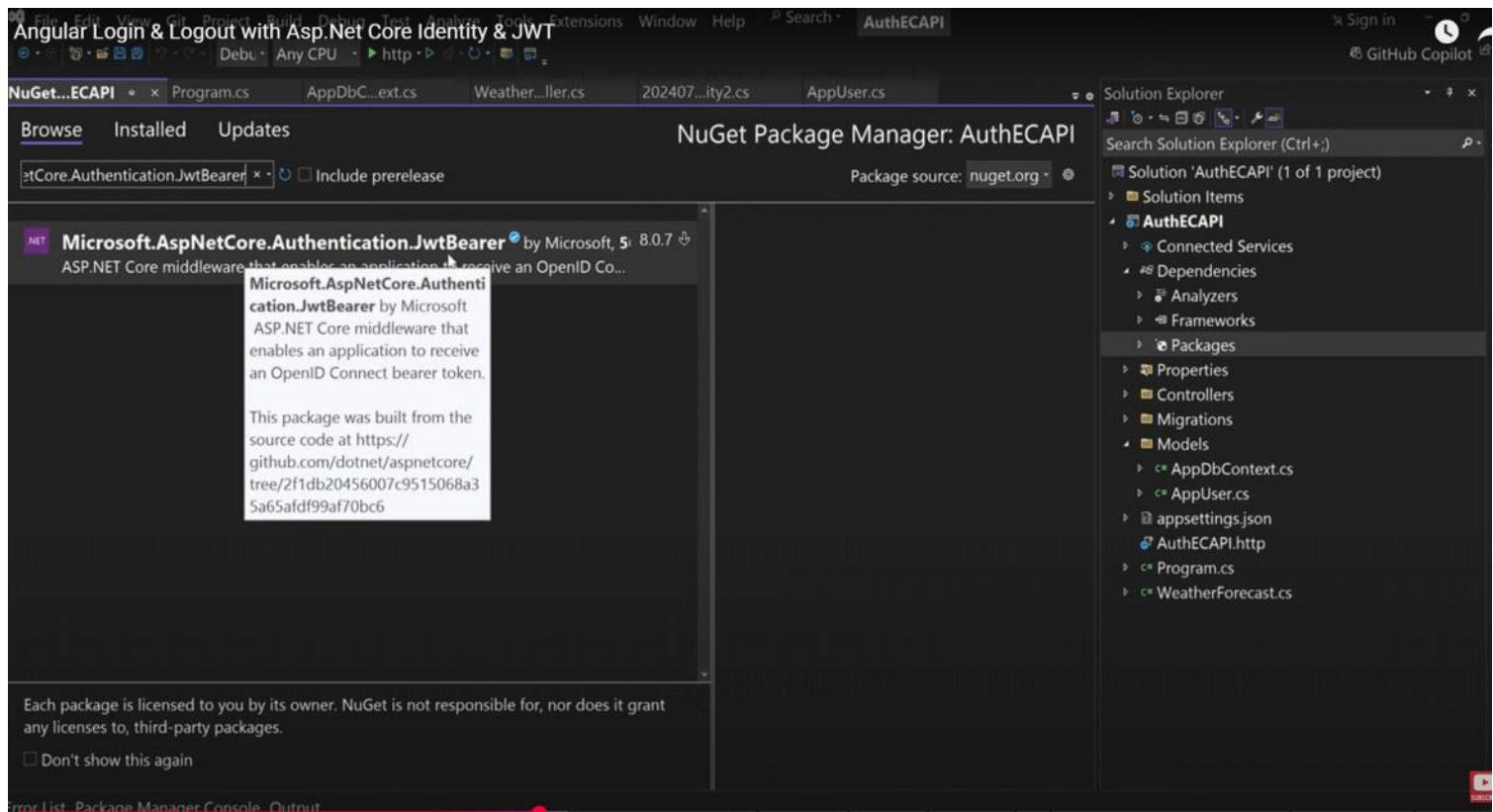
Payload

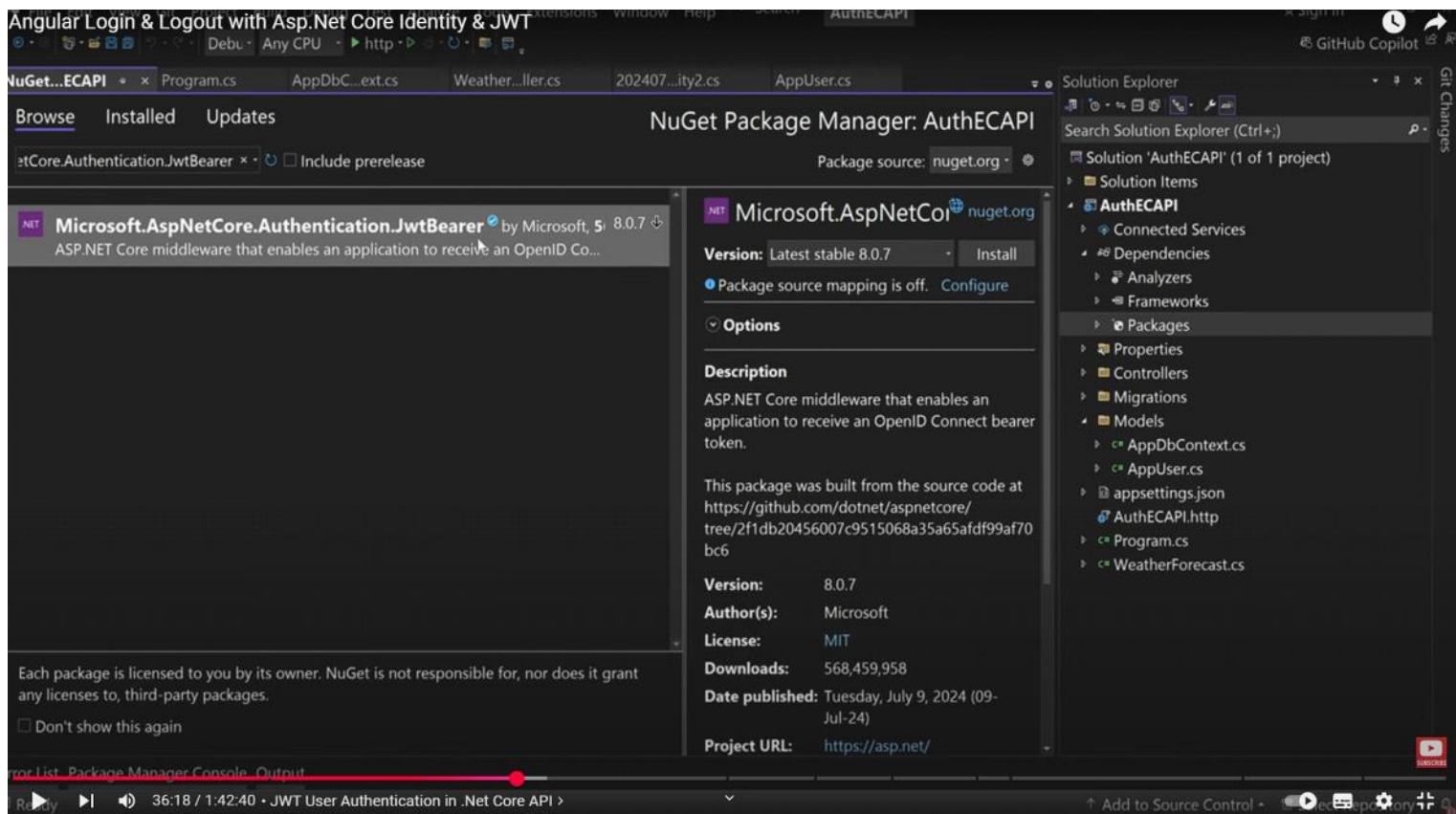
Then, this JSON is **Base64Url** encoded to form the first part of the JWT.

Payload

The second part of the token is the payload, which contains the claims. **Claims are statements about an entity** (typically, the user) and additional data. There are three types of claims: *registered*, *public*, and *private* claims.

- **Registered claims:** These are a set of predefined claims which are not mandatory but recommended, to provide a set of useful, interoperable claims. Some of them are: **iss** (issuer), **exp** (expiration time), **sub** (subject), **aud** (audience), and **others**.





```
Angular Login & Logout with Asp.Net Core Identity & JWT
```

```
options.User.RequireUniqueEmail = true;
});

builder.Services.AddDbContext<AppDbContext>(options =>
    options.UseSqlServer(builder.Configuration.GetConnectionString("DevDB"));

builder.Services.AddAuthentication(x =>
{
    x.DefaultAuthenticateScheme =
    x.DefaultChallengeScheme =
    x.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
});

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}
```

Solution Explorer

Solution 'AuthECAPI' (1 of 1 project)

AuthECAPI

- Connected Services
- Dependencies
- Analyzers
- Frameworks
- Packages
 - Properties
 - Controllers
 - Migrations
 - Models
 - AppDbContext.cs
 - AppUser.cs
 - appsettings.json
 - AuthECAPI.http
 - Program.cs
 - WeatherForecast.cs

Angular Login & Logout with Asp.Net Core Identity & JWT

```
program.cs*  AppDbC...ext.cs  Weather...ller.cs  202407...ity2.cs  ApplicationUser.cs  appset...gs.json
```

AuthECAPI

```
    x.DefaultChallengeScheme =
        x.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
    ).AddJwtBearer(y =>
{
    y.SaveToken = false;
    y.TokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuerSigningKey = true,
        IssuerSigningKey = new SymmetricSecurityKey(
            Encoding.UTF8.GetBytes()
        );
};

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}
```

>Config.CORS

Package Manager Console Output

40:08 / 1:42:40 • JWT User Authentication in .Net Core API

Solution Explorer

- Search Solution Explorer (Ctrl+Shift+F)
- Solution 'AuthECAPI' (1 of 1 project)
 - AuthECAPI
 - Connected Services
 - Dependencies
 - Properties
 - Controllers
 - Migrations
 - Models
 - AppDbContext.cs
 - AppUser.cs
 - appsettings.json
 - AuthECAPI.http
 - Program.cs
 - WeatherForecast.cs

GitHub Copilot

Git Changes

schema: https://json.schemastore.org/appsettings.json

```
appset...gs.json*
```

```
"Default": "Information",
"Microsoft.AspNetCore": "Warning"
},
"AllowedHosts": "*",
"ConnectionStrings": {
    "DevDB": "Server=(local)\\sqlexpress; Database=AuthEDB; Trusted_Connection=True"
},
"AppSettings": {
    "JWTSecret": "GiveASecretKeyHavingAtleast32Characters"
}
```

Solution Explorer

- Search Solution Explorer (Ctrl+Shift+F)
- Solution 'AuthECAPI' (1 of 1 project)
 - AuthECAPI
 - Connected Services
 - Dependencies
 - Properties
 - Controllers
 - Migrations
 - Models
 - AppDbContext.cs
 - AppUser.cs
 - appsettings.json
 - AuthECAPI.http
 - Program.cs
 - WeatherForecast.cs

GitHub Copilot

Git Changes

The screenshot shows the Visual Studio IDE interface. The top navigation bar includes tabs for 'Program.cs*', 'AppDbC...ext.cs', 'Weather...ller.cs', '202407...ity2.cs', 'AppUser.cs', and 'appset...gs.json'. Below the tabs, the main code editor displays the 'Program.cs' file with the following content:

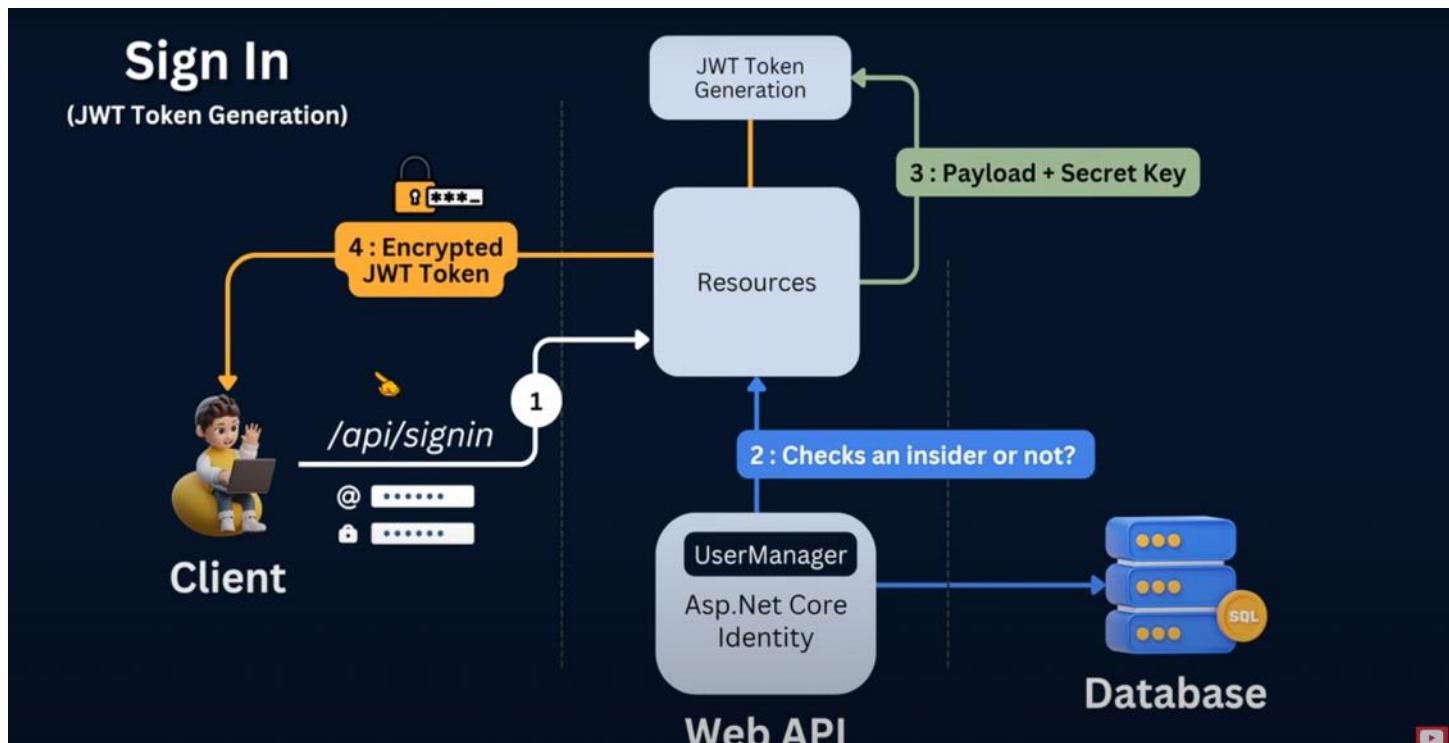
```
AuthECAPI
    x.DefaultChallengeScheme =
        x.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
}).AddJwtBearer(y =>
{
    y.SaveToken = false;
    y.TokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuerSigningKey = true,
        IssuerSigningKey = new SymmetricSecurityKey(
            Encoding.UTF8.GetBytes(
                builder.Configuration["AppSettings:JWTSecret"]!))
    };
});
builder.Build();

// the HTTP request pipeline.
Environment.IsDevelopment()

Swagger();
SwaggerUI();
```

To the right of the code editor is the 'Solution Explorer' window, which lists the project structure for 'AuthECAPI' (1 item). The project contains the following files:

- Solution 'AuthECAPI' (1 item)
- Solution Items
- AuthECAPI
 - Connected Services
 - Dependencies
 - Properties
 - Controllers
 - Migrations
 - Models
 - AppDbContext.cs
 - AppUser.cs
 - appsettings.json
 - AuthECAPI.http
 - Program.cs
 - WeatherForecast.cs



In programme.cs

```
~public class LoginModel
{
    public string Email { get; set; }
    public string Password { get; set; }
}
```

The screenshot shows the Visual Studio IDE interface. On the left, the code editor displays `Program.cs` with the following content:

```
app.MapPost(...);

app.MapPost("/api/signin", async (
    UserManager<AppUser> userManager,
    [FromBody] LoginModel loginModel) =>
{
    var user = await userManager.FindByEmailAsync(loginModel.Email);
    if (user != null && await userManager.CheckPasswordAsync(user, loginModel.Password))
    {
        var signInKey = new SymmetricSecurityKey(
            Encoding.UTF8.GetBytes(builder.Configuration["AppSettings:JWTSecret"]!));
        var tokenDescriptor = new SecurityTokenDescriptor
        {
            Subject = new ClaimsIdentity(new Claim[]
            {
                new Claim("UserID", user.Id.ToString())
            }),
            Expires = DateTime.UtcNow.AddMinutes()
        };
    }
    else
        return Results.BadRequest(new { message = "Username or password is incorrect." });
});
```

On the right, the Solution Explorer window is open, showing the project structure for 'AuthECAPI'. The project contains the following files:

- Solution Items
- AuthECAPI
 - Connected Services
 - Dependencies
 - Analyzers
 - Frameworks
 - Packages
 - Properties
 - Controllers
 - Migrations
 - Models
 - AppDbContext.cs
 - AppUser.cs
 - appsettings.json
 - AuthECAPI.http
 - Program.cs
 - WeatherForecast.cs

The screenshot shows the Visual Studio IDE with the AuthECAP1 project open. The Solution Explorer on the right lists files like Program.cs, AppDbContext.cs, ApplicationUser.cs, appsettings.json, AuthECAP1.http, and WeatherForecast.cs. The code editor on the left contains C# code for generating a JWT token. It uses the UserManager to find a user by email and checks the password. It then creates a SymmetricSecurityKey from the app settings, initializes a SecurityTokenDescriptor with claims (User ID), expiration, and signing credentials (HmacSha256Signature), and finally writes the token to a JwtSecurityTokenHandler.

```

UserManager< ApplicationUser > userManager,
[FromBody] LoginModel loginModel) =>
{
    var user = await userManager.FindByEmailAsync(loginModel.Email);
    if (user != null && await userManager.CheckPasswordAsync(user, loginModel.Password))
    {
        var signInKey = new SymmetricSecurityKey(
            Encoding.UTF8.GetBytes(builder.Configuration["AppSettings:JWTSecret"]!));
        var tokenDescriptor = new SecurityTokenDescriptor
        {
            Subject = new ClaimsIdentity(new Claim[]
            {
                new Claim("UserID", user.Id.ToString())
            }),
            Expires = DateTime.UtcNow.AddMinutes(10),
            SigningCredentials = new SigningCredentials(
                signInKey,
                SecurityAlgorithms.HmacSha256Signature
            )
        };
        var tokenHandler = new JwtSecurityTokenHandler();
        var securityToken = tokenHandler.CreateToken(tokenDescriptor);
        var token = tokenHandler.WriteToken(securityToken);
        return Results.Ok(new {token});
    }
    else
        return Results.BadRequest(new { message = "Username or password is incorrect." });
};

app.Run();

```

The screenshot shows the jwt.io debugger interface. The token input field contains a JWT token. The 'Encoded' section shows the raw token string. The 'Decoded' section shows the parsed JSON structure with fields like 'alg' (HS256), 'typ' (JWT), and a payload containing 'User ID', 'nbf', 'exp', and 'iat'. The 'VERIFY SIGNATURE' section shows the HMACSHA256 verification logic using the base64 URL encoded header and payload plus the secret key.

Encoded: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJvc2VySUpiOjI3NmQ5Mzk1Ny03NzdmLTQ8MmUtYTRiY1010DB1ZWIZMGQyYTM1LCJuYmY10je3MjE4MjM2MDgsImV4cCI6MTcyMjY4NzYwOCwiaWF0IjoxNzIxODIzNjA4fQ.3Hzfdz1D2vB3Q3EQpttmZo2XGpmp6zkNa0XTRahwWtY

Decoded:

```

HEADER: ALGORITHM & TOKEN TYPE
{
  "alg": "HS256",
  "typ": "JWT"
}

PAYLOAD: DATA
{
  "User ID": "76d93957-777f-442e-a4bb-580eeb38d2a3",
  "nbf": 1721823688,
  "exp": 1722687688,
  "iat": 1721823688
}

VERIFY SIGNATURE
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
)

```

```
Angular Login & Logout with Asp.Net Core Identity & JWT
```

```
AuthECClient
```

```
login.component.ts auth.service.ts registration.component.ts login.component.html registration.component.html user.component.html
```

```
export class AuthService {
```

```
    constructor(private http:HttpClient) { }
```

```
    baseURL = 'http://localhost:5007/api';
```

```
    createUser(formData:any){
```

```
        return this.http.post(this.baseURL+'/signup',formData);
```

```
}
```

```
    signin(formData:any){
```

```
        return this.http.post(this.baseURL+'/signin',formData);
```

```
}
```

```
}
```

Service class

```
Angular Login & Logout with Asp.Net Core Identity & JWT
```

```
AuthECClient
```

```
login.component.ts auth.service.ts registration.component.ts login.component.html registration.component.html user.component.html
```

```
}
```

```
export class LoginComponent {
```

```
    constructor(
```

```
        public formBuilder: FormBuilder,
```

```
        private service:AuthService) { }
```

```
    isSubmitted: boolean = false;
```

```
    form = this.formBuilder.group({
```

```
        email: ['', Validators.required],
```

```
        password: ['', Validators.required],
```

```
    })
```

```
    hasDisplayableError(controlName: string): Boolean {
```

```
        const control = this.form.get(controlName);
```

```
        return Boolean(control?.invalid) &&
```

```
            (this.isSubmitted || Boolean(control?.touched) || Boolean(control?.dirty))
```

```
}
```

Angular Login & Logout with Asp.Net Core Identity & JWT

```
const control = this.form.get(controlName);
return Boolean(control?.invalid) &&
   (this.isSubmitted || Boolean(control?.touched) || Boolean(control?.dirty))
}

onSubmit(){
  this.isSubmitted = true;
  if(this.form.valid){
    this.service.signin(this.form.value).subscribe({
      next:(res:any)=>{
        localStorage.setItem('token',res.token);
      },
      error:err=>{}
    })
  }
}
```

Angular Login & Logout with Asp.Net Core Identity & JWT

```
EXPLORER: AUTHECCLIENT ... app.routes.ts X login.component.ts dashboard.component.ts auth.service.ts registration.compo ...
public favicon.ico
src
app
  dashboard
    dashboard.component.html
    dashboard.component.ts
  shared
  user
    login
      login.component.html
      login.component.ts
    registration
      registration.component.html
      registration.component.ts
      user.component.html
      user.component.ts
      app.component.html
      app.component.ts
      app.config.ts
    app.routes.ts
    index.html
    main.ts
```

```
import { Routes } from '@angular/router';
import { UserComponent } from './user/user.component';
import { RegistrationComponent } from './user/registration/registration.component';
import { LoginComponent } from './user/login/login.component';
import { DashboardComponent } from './dashboard/dashboard.component';

export const routes: Routes = [
  {path: '', component:UserComponent,
  children:[
    {path:'signup',component:RegistrationComponent},
    {path:'signin',component:LoginComponent},
  ]
},
  {path:'dashboard',component:DashboardComponent}
];
```

EXPLORER: AUTHECLIENT

... app.component.html app.routes.ts login.component.ts 2 X dashboard.component.ts auth.service.ts ...

```
styles: ``

}

export class LoginComponent {
  constructor(
    public formBuilder: FormBuilder,
    private service: AuthService,
    private router: Router) { }

  isSubmitted(): boolean {
    return this.form.valid;
  }

  hasError(controlName: string): boolean {
    const control = this.form.get(controlName);
    return Boolean(control?.invalid) && control.touched;
  }
}
```

Angular Login & Logout with Asp.Net Core Identity & JWT

AuthEClient

EXPLORER: AUTHECLIENT

... component.html app.routes.ts login.component.ts X dashboard.component.ts auth.service.ts rec ...

```
}
```

```
onSubmit(): void {
  this.isSubmitted = true;
  if (this.form.valid) {
    this.service.signin(this.form.value).subscribe({
      next: (res: any) => {
        localStorage.setItem('token', res.token);
        this.router.navigateByUrl('/dashboard');
      },
      error: err => {
        console.error(err);
      }
    });
  }
}
```

Angular Login & Logout with Asp.Net Core Identity & JWT

AuthECCClient

EXPLORER: AUTHCLIENT

component.html app.routes.ts login.component.ts 1 dashboard.component.ts auth.service.ts

```
imports: [CommonModule, ReactiveFormsModule, RouterLink],
templateUrl: './login.component.html',
styles: ``

}

export class LoginComponent {
constructor(
  public formBuilder: FormBuilder,
  private service: AuthService,
  private router: Router,
  private toastr: ToastrService) { }
isSubmitted: boolean = false;

form = this.formBuilder.group({
  email: ['', Validators.required],
  password: ['', Validators.required],
})
}
```

component.html app.routes.ts login.component.ts 1 dashboard.component.ts auth.service.ts registration.component.ts login.comp ...

```
this.isSubmitted = true;
if (this.form.valid) {
  this.service.signin(this.form.value).subscribe({
    next: (res: any) => {
      localStorage.setItem('token', res.token);
      this.router.navigateByUrl('/dashboard');
    },
    error: err => {
      if (err.status == 400)
        this.toastr.error('Incorrect email or password.', 'Login failed')
      else
        console.log('error during login:\n', err);
    }
  })
}
```

Angular Login & Logout with Asp.Net Core Identity & JWT

EXPLORER: AUTHECLIENT

```
registration.component.html user.component.html user.component.ts 1 app.config.ts app.compon ...
```

src
app
user
 login
 login.component.ts
 registration
 registration.component.html
 registration.component.ts
 user.component.html
 user.component.ts
 app.component.html
 app.component.ts
 app.config.ts
 app.routes.ts
 index.html
 main.ts
 styles.css
.editorconfig
.gitignore
angular.json
app-structure.txt
package-lock.json
package.json

```
import { RouterOutlet } from '@angular/router';
import { trigger, style, animate, transition, query } from '@angular/animations';

@Component({
  selector: 'app-user',
  standalone: true,
  imports: [RegistrationComponent, RouterOutlet],
  templateUrl: './user.component.html',
  styles: ``,
  animations: [
    trigger('routerFadeIn', [
      transition('* <-> *', [
        query(':enter', [
          style({ opacity: 0 })
        ], { optional: true })
      ])
    ])
  ]
})
export class UserComponent {
```

10:54 1:42:40 • Angular Animation >

Angular Login & Logout with Asp.Net Core Identity & JWT

EXPLORER: AUTHECLIENT

```
registration.component.html user.component.html user.component.ts X app.config.ts app.compon ...
```

src
app
user
 login
 login.component.ts
 registration
 registration.component.html
 registration.component.ts
 user.component.html
 user.component.ts
 app.component.html
 app.component.ts
 app.config.ts
 app.routes.ts
 index.html
 main.ts
 styles.css
.editorconfig
.gitignore
angular.json
app-structure.txt
package-lock.json
package.json

```
@Component({
  selector: 'app-user',
  standalone: true,
  imports: [RegistrationComponent, RouterOutlet],
  templateUrl: './user.component.html',
  styles: ``,
  animations: [
    trigger('routerFadeIn', [
      transition('* <-> *', [
        query(':enter', [
          style({ opacity: 0 })
        ], { optional: true })
      ])
    ])
  ]
})
export class UserComponent {
```

Angular Login & Logout with Asp:Net Core Identity & JWT

EXPLORER: AUTHECCLIENT

registration.component.html user.component.html 1 user.component.ts app.config.ts app.compon ...

```
<div class="col-xxl-6 offset-xxl-3 col-lg-8 offset-lg-2 col-10 offset-1">
  <div class="row rounded-4 border-0 shadow-lg bg-white" style="min-height:450px;">
    <div class="col-md-5 d-md-flex d-none bg-success rounded-start-4 align-items-center justify-content-center">
      <div class="text-center text-white">
        
        <h2 class="fw-bolder">Auth EC</h2>
      </div>
    </div>
    <div class="col-md-7 col-12 d-flex justify-content-start align-items-center">
      <div class="p-5 w-100" [@routerFadeIn]="getRouteUrl()">
        <router-outlet />
      </div>
    </div>
  </div>
</div>
```

Angular Login & Logout with Asp:Net Core Identity & JWT

EXPLORER: AUTHECCLIENT

registration.component.html user.component.html user.component.ts app.config.ts app.compon ...

```
query(':enter', [
  style({ opacity: 0 }),
  animate('is ease-in-out', style({ opacity: 1 })),
], { optional: true })
])
]
})
}

export class UserComponent {

  constructor(private context: ChildrenOutletContexts){}

  getRouteUrl() {
    return this.context.getContext('primary')
  }
}
```

The screenshot shows an IDE interface with the following details:

- EXPLORER: AUTHECLIENT**: Shows the project structure with files like `src/app/user/login/login.component.ts`, `src/app/user/registration/registration.component.ts`, `src/app/app.component.ts`, etc.
- registration.component.html**: A template file with Angular structural directives (`ngIf`, `ngFor`) and styles.
- user.component.html**: A template file with Angular structural directives and styles.
- user.component.ts**: The active file, showing the implementation of the `UserComponent`. It includes a CSS query for entering states and an `Observable` for route URLs.
- app.config.ts**: Configuration file.
- app.component.ts**: Main application component.

```
query(':enter', [
    style({ opacity: 0 }),
    animate('1s ease-in-out', style({ opacity: 1 })),
], { optional: true }),
])
])
}
)
}

export class UserComponent {
    (method) UserComponent.getRouteUrl(): Observable<UrlSegment[]> | undefined
    getRouteUrl(): {
        ...return this.context.getContext('primary')?.route?.url;
        ...
    }
}
```

Angular Login & Logout with Asp:Net Core Identity & JWT

AuthECClient

EXPLORER: AUTHECLIENT

registration.component.html user.component.html user.component.ts app.config.ts app.component.ts

```
<div class="col-xxl-6 offset-xxl-3 col-lg-8 offset-lg-2 col-10 offset-1">
  <div class="row rounded-4 border-0 shadow-lg bg-white" style="min-height:450px;">
    <div class="col-md-5 d-md-flex d-none bg-success rounded-start-4 align-items-center justify-content-center">
      <div class="text-center text-white">
        
        <h2 class="fw-bolder">Auth EC</h2>
      </div>
    </div>
    <div class="col-md-7 col-12 d-flex justify-content-start align-items-center">
      <div class="p-5 w-100" [@routerFadeIn]="getRouteUrl()">
        <router-outlet />
      </div>
    </div>
  </div>
</div>
```

EXPLORER: AUTHECLIENT

```
src
  app
    user
      login
        login.component.ts
      registration
        registration.component.html
        registration.component.ts
        user.component.html
        user.component.ts
      app.component.html
      app.component.ts
      app.config.ts
      app.routes.ts
    index.html
    main.ts
    styles.css
.editorconfig
.gitignore
angular.json
app-structure.txt
package-lock.json
package.json
```

app.routes.ts

```
import { Routes } from '@angular/router';
import { UserComponent } from './user/user.component';
import { RegistrationComponent } from './user/registration/registration.component';
import { LoginComponent } from './user/login/login.component';
import { DashboardComponent } from './dashboard/dashboard.component';

export const routes: Routes = [
  { path: '', redirectTo: '/signin', pathMatch: 'full' },
  {
    path: '',
    component: UserComponent,
    children: [
      { path: 'signup', component: RegistrationComponent },
      { path: 'signin', component: LoginComponent },
    ]
  },
  { path: 'dashboard', component: DashboardComponent }
];
```

EXPLORER: AUTHECLIENT

```
public
src
  app
    dashboard
      dashboard.component.html 1
      dashboard.component.ts
    shared
    user
      login
        login.component.html
        login.component.ts
    registration
      registration.component.html
      registration.component.ts
      user.component.html
      user.component.ts
    app.component.html
    app.component.ts
    app.config.ts
    app.routes.ts
  index.html
  main.ts
```

dashboard.component.html 1

```
<button class="btn btn-dark" (click)="onLogout()">Logout</button>
```

EXPLORER: AUTHECLIENT

app.routes.ts dashboard.component.html login.component.ts dashboard.component.ts auth...

```
@Component({
  selector: 'app-dashboard',
  standalone: true,
  imports: [],
  templateUrl: './dashboard.component.html',
  styles: []
})
export class DashboardComponent {

  constructor(private router: Router) { }

  onLogout() {
    localStorage.removeItem('token');
    this.router.navigateByUrl('/signin');
  }
}
```

C# Extension Methods

Framework ABC

```
namespace ThirdParty
{
    public interface/class Original
    {
        // Owned Methods & Properties.
    }
}
```

A Seperate File

```
namespace Extensions
{
    public static class MeaningfulName
    {
        public static void ExtraMethod1(this Original org)
        {...}

        public static void ExtraMethod2(this Original org)
        {...}
    }
}
```

Program.cs

```
...
//Say org1 is of the type Original
org1.ExtraMethod1();
org1.ExtraMethod2();
```

Angular Login & Logout with Asp.Net Core Identity & JWT

The screenshot shows the Visual Studio IDE interface. On the left, the code editor displays `Program.cs` with C# code for setting up an ASP.NET Core application with Identity and JWT authentication. On the right, the Solution Explorer pane shows the project structure for 'AuthECAPI'.

```
using AuthECAPI.Models;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.IdentityModel.Tokens;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

// Services from Identity Core.
builder.Services
    .AddIdentityApiEndpoints<AppUser>()
    .AddEntityFrameworkStores<AppDbContext>();
```

Solution Explorer content:

- Solution 'AuthECAPI' (1 of 1 project)
 - AuthECAPI
 - Connected Services
 - Dependencies
 - Properties
 - Controllers
 - Migrations
 - Models
 - Extensions
 - appsettings.json
 - AuthECAPI.csproj
 - Program.cs
 - WeatherForecast.cs

Angular Login & Logout With Asp.net Core Identity - 6.1

```

AuthECAPISolution Explorer
Program.cs
AuthECAPI
AuthECAPI.Models;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.IdentityModel.Tokens;
using System.IdentityModel.Tokens.Jwt;
using System.Text;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

// Services from Identity Core.
builder.Services
    .AddIdentityApiEndpoints<AppUser>()
    .AddEntityFrameworkStores<AppDbContext>();

// Add services to the container.

builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

// Services from Identity Core.
builder.Services
    .AddIdentityApiEndpoints<AppUser>()
    .AddEntityFrameworkStores<AppDbContext>();

builder.Services.AddIdentityHandlersAndStores();

builder.Services.Configure<IdentityOptions>(options =>
{
    options.Password.RequireDigit = false;
    options.Password.RequireUppercase = false;
    options.Password.RequireLowercase = false;
    options.User.RequireUniqueEmail = true;
});

builder.Services.AddDbContext<AppDbContext>(options =>
    options.UseSqlServer(builder.Configuration.GetConnectionString("DevDB")));

builder.Services.AddAuthentication(x =>
{
    x.AddJwtBearer("Auth", options =>
    {
        options.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuer = false,
            ValidateAudience = false,
            ValidateLifetime = true,
            ValidateIssuerSigningKey = true,
            IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("Your Secret Key"))
        };
    });
});

```

No issues found

Add New Item - AuthECAPI

Search Solution Explorer (Ctrl+.)

Solution 'AuthECAPI' (1 of 1 project)

Solution Items

AuthECAPI

- Connected Services
- Dependencies
- Properties
- Controllers
- Extensions
- Migrations
- Models
- appsettings.json
- AuthECAPI.http
- Program.cs
- WeatherForecast.cs

Name: IdentityExtensions

Add Cancel

Line 19 Ch 1 SPC CRLF

AuthECAPISolution Explorer
Program.cs
AuthECAPI
AuthECAPI.Models;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.IdentityModel.Tokens;
using System.IdentityModel.Tokens.Jwt;
using System.Text;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

// Services from Identity Core.
builder.Services
 .AddIdentityApiEndpoints<AppUser>()
 .AddEntityFrameworkStores<AppDbContext>();

// Add services to the container.

builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

// Services from Identity Core.
builder.Services
 .AddIdentityApiEndpoints<AppUser>()
 .AddEntityFrameworkStores<AppDbContext>();

builder.Services.AddIdentityHandlersAndStores();

builder.Services.Configure<IdentityOptions>(options =>
{
 options.Password.RequireDigit = false;
 options.Password.RequireUppercase = false;
 options.Password.RequireLowercase = false;
 options.User.RequireUniqueEmail = true;
});

builder.Services.AddDbContext<AppDbContext>(options =>
 options.UseSqlServer(builder.Configuration.GetConnectionString("DevDB")));

builder.Services.AddAuthentication(x =>
{
 x.AddJwtBearer("Auth", options =>
 {
 options.TokenValidationParameters = new TokenValidationParameters
 {
 ValidateIssuer = false,
 ValidateAudience = false,
 ValidateLifetime = true,
 ValidateIssuerSigningKey = true,
 IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("Your Secret Key"))
 };
 });
});

// Services from Identity Core.
builder.Services
 .AddIdentityApiEndpoints<AppUser>()
 .AddEntityFrameworkStores<AppDbContext>();

// Add services to the container.

builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

// Services from Identity Core.
builder.Services
 .AddIdentityApiEndpoints<AppUser>()
 .AddEntityFrameworkStores<AppDbContext>();

builder.Services.AddIdentityHandlersAndStores();

builder.Services.Configure<IdentityOptions>(options =>
{
 options.Password.RequireDigit = false;
 options.Password.RequireUppercase = false;
 options.Password.RequireLowercase = false;
 options.User.RequireUniqueEmail = true;
});

builder.Services.AddDbContext<AppDbContext>(options =>
 options.UseSqlServer(builder.Configuration.GetConnectionString("DevDB")));

builder.Services.AddAuthentication(x =>
{
 x.AddJwtBearer("Auth", options =>
 {
 options.TokenValidationParameters = new TokenValidationParameters
 {
 ValidateIssuer = false,
 ValidateAudience = false,
 ValidateLifetime = true,
 ValidateIssuerSigningKey = true,
 IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("Your Secret Key"))
 };
 });
});

No issues found

Search Solution Explorer (Ctrl+.)

Solution 'AuthECAPI' (1 of 1 project)

Solution Items

AuthECAPI

- Connected Services
- Dependencies
- Properties
- Controllers
- Extensions
- Migrations
- Models
- appsettings.json
- AuthECAPI.http
- Program.cs
- WeatherForecast.cs

Line 19 Ch 1 SPC CRLF

Cut that

Angular Login & Logout With Asp.Net Core Identity - A SPA

Solution Explorer

Search Solution Explorer (Ctrl+;) Solution 'AuthECAPI' (1 of 1 project)

- AuthECAPI
 - Connected Services
 - Dependencies
 - Properties
 - Controllers
 - Extensions
 - IdentityExtensions.cs
 - Migrations
 - Models
 - appsettings.json
 - AuthECAPI.http
 - Program.cs
 - WeatherForecast.cs

Program.cs

```
// Add services to the container.

builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

// Services from Identity Core.
builder.Services
    .AddIdentityApiEndpoints<AppUser>()
    .AddEntityFrameworkStores<AppDbContext>();

builder.Services.AddIdentityHandlersAndStores();

builder.Services.Configure<IdentityOptions>(options =>
{
    options.Password.RequireDigit = false;
    options.Password.RequireUppercase = false;
    options.Password.RequireLowercase = false;
    options.User.RequireUniqueEmail = true;
});

builder.Services.AddDbContext<AppDbContext>(options =>
    options.UseSqlServer(builder.Configuration.GetConnectionString("DevDB")));

builder.Services.AddAuthentication(x =>
```

No issues found

Output

Angular Login & Logout With Asp.Net Core Identity - A SPA

Solution Explorer

Search Solution Explorer (Ctrl+;) Solution 'AuthECAPI' (1 of 1 project)

- AuthECAPI
 - Connected Services
 - Dependencies
 - Properties
 - Controllers
 - Extensions
 - IdentityExtensions.cs
 - Migrations
 - Models
 - appsettings.json
 - AuthECAPI.http
 - Program.cs
 - WeatherForecast.cs

IdentityExtensions.cs

```
using AuthECAPI.Models;

namespace AuthECAPI.Extensions
{
    public static class IdentityExtensions
    {
        public static void AddIdentityHandlersAndStores(this IServiceCollection services)
        {
            services.AddIdentityApiEndpoints<AppUser>()
                .AddEntityFrameworkStores<AppDbContext>();
        }
    }
}
```

No issues found

Output

Cut this
Into

The screenshot shows the Visual Studio IDE interface with the following details:

- File Bar:** File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, Full Screen.
- Solution Explorer:** Shows the solution 'AuthECAPI' (1 of 1 project) containing the 'AuthECAPI' project with files like Connected Services, Dependencies, Properties, Controllers, Extensions (IdentityExtensions.cs), Migrations, Models, appsettings.json, Program.cs, and WeatherForecast.cs.
- Code Editor:** The 'Program.cs' file is open, showing the configuration of services. It includes code for adding controllers, endpoints, swagger, identity handlers, and stores. A specific section for configuring IdentityOptions is highlighted, showing the disabling of password requirements (Digit, Uppercase, Lowercase) and enabling unique email requirement.
- Status Bar:** Shows 'No Issues found' and other status indicators.
- Taskbar:** Shows the current file is 'Organize Program.cs File'.

Angular Login & Logout with Asp.Net Core Identity & JWT

```
AuthECAPI.cs* Program.cs* %AuthECAPI.Extensions.IdentityExtensions ConfigureIdentityOptions(IServiceCollection services)
```

```
AuthECAPI
+ using AuthECAPI.Models;
+ using Microsoft.AspNetCore.Identity;

namespace AuthECAPI.Extensions
{
    public static class IdentityExtensions
    {
        public static void AddIdentityHandlersAndStores(this IServiceCollection services)
        {
            services.AddIdentityApiEndpoints<AppUser>()
                .AddEntityFrameworkStores<AppDbContext>();
        }

        public static void ConfigureIdentityOptions(this IServiceCollection services)
        {
            services.Configure<IdentityOptions>(options =>
            {
                options.Password.RequireDigit = false;
                options.Password.RequireUppercase = false;
                options.Password.RequireLowercase = false;
                options.User.RequireUniqueEmail = true;
            });
        }
    }
}
```

Solution Explorer

- Search Solution Explorer (Ctrl+Shift+F)
- Solution 'AuthECAPI' (1 of 1 project)
 - Solution Items
 - AuthECAPI
 - Connected Services
 - Dependencies
 - Properties
 - Controllers
 - Extensions
 - IdentityExtensions.cs
 - Migrations
 - Models
 - appsettings.json
 - AuthECAPI.http
 - Program.cs
 - WeatherForecast.cs

Angular Login & Logout with Asp.Net Core Identity & JWT

```
IdentityExtensions.cs Program.cs* %UserRegistrationModel Email
```

```
// Add services to the container.

builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

builder.Services.AddIdentityHandlersAndStores();
builder.Services.ConfigureIdentityOptions();

builder.Services.AddDbContext<AppDbContext>(options =>
    options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));

builder.Services.AddAuthentication(x =>
{
    x.DefaultAuthenticateScheme =
    x.DefaultChallengeScheme =
    x.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
})
```

Solution Explorer

- Search Solution Explorer (Ctrl+Shift+F)
- Solution 'AuthECAPI' (1 of 1 project)
 - Solution Items
 - AuthECAPI
 - Connected Services
 - Dependencies
 - Properties
 - Controllers
 - Extensions
 - IdentityExtensions.cs
 - Migrations
 - Models
 - appsettings.json
 - AuthECAPI.http
 - Program.cs
 - WeatherForecast.cs

The screenshot shows the Visual Studio IDE with the solution 'AuthECAPI' open. The left pane displays the code for `IdentityExtensions.cs` in the `AuthECAPI.Extensions` namespace. The right pane shows the Solution Explorer with the project structure:

- Solution Items
- AuthECAPI
 - Connected Services
 - Dependencies
 - Properties
 - Controllers
 - Extensions
 - IdentityExtensions.cs
 - Migrations
 - Models
 - appsettings.json
 - AuthECAPI.http
 - Program.cs
 - WeatherForecast.cs

The screenshot shows the Visual Studio IDE with the solution 'AuthECAPI' open. The left pane displays the code for `Program.cs`. The right pane shows the Solution Explorer with the project structure:

- Solution Items
- AuthECAPI
 - Connected Services
 - Dependencies
 - Properties
 - Controllers
 - Extensions
 - IdentityExtensions.cs
 - Migrations
 - Models
 - appsettings.json
 - AuthECAPI.http
 - Program.cs
 - WeatherForecast.cs

A screenshot of the Visual Studio IDE interface. The main window shows the code for `Program.cs` in the `AuthECAPI` project. A context menu is open over the code editor, specifically over the line `builder.Services.AddControllers();`. The menu is titled "Add New Item - AuthECAPI". It lists several item types under "C#": Class, Interface, Code File, General, ASP.NET Core, Code, Data, General, Web, and CSharp. The "Type: C#" dropdown is set to "Class". A tooltip for "Class" says "An empty class declaration". The "Name:" field contains "EFCoreExtension". At the bottom right of the dialog are "Add" and "Cancel" buttons.

```
// Add services to the container.

builder.Services.AddControllers()
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

builder.Services.AddIdentityHandlersAndStores()
    .ConfigureIdentityOptions();

builder.Services.AddDbContext<AppDbContext>(options =>
    options.UseSqlServer(builder.Configuration.GetConnectionString("DevDB")));

builder.Services.AddAuthentication(x =>
{
    x.DefaultAuthenticateScheme =
    x.DefaultChallengeScheme =
    x.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
}).AddJwtBearer(y =>
{
    y.SaveToken = false;
    y.TokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuerSigningKey = true,
        IssuerSigningKey = new SymmetricSecurityKey(

```

A screenshot of the Visual Studio IDE interface. The main window shows the code for `Program.cs` in the `AuthECAPI` project. The "Extensions" folder in the Solution Explorer contains the newly created `EFCoreExtension.cs` file. The code editor shows the same code as the previous screenshot, with the `EFCoreExtension.cs` file now listed in the Solution Explorer under the `AuthECAPI` project.

```
// Add services to the container.

builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

builder.Services.AddIdentityHandlersAndStores()
    .ConfigureIdentityOptions();

builder.Services.AddDbContext<AppDbContext>(options =>
    options.UseSqlServer(builder.Configuration.GetConnectionString("DevDB")));

builder.Services.AddAuthentication(x =>
{
    x.DefaultAuthenticateScheme =
    x.DefaultChallengeScheme =
    x.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
}).AddJwtBearer(y =>
{
    y.SaveToken = false;
    y.TokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuerSigningKey = true,
        IssuerSigningKey = new SymmetricSecurityKey(

```

Angular Login & Logout with Asp.Net Core Identity & JWT

```
EFCore...ions.cs  Program.cs  InjectDbContext(IServiceCollection services, ICor...
```

```
AuthECAPI
```

```
AuthECAPI.Models;
using Microsoft.EntityFrameworkCore;
```

```
namespace AuthECAPI.Extensions
{
    public static class EFCoreExtensions
    {
        public static IServiceCollection InjectDbContext(
            this IServiceCollection services,
            IConfiguration config)
        {
            services.AddDbContext<AppDbContext>(options =>
                options.UseSqlServer(config.GetConnectionString("DevDB")));
            return services;
        }
    }
}
```

```
Solution Explorer
Search Solution Explorer (Ctrl+.)
Solution 'AuthECAPI' (1 of 1 project)
AuthECAPI
Connected Services
Dependencies
Properties
Controllers
Extensions
EFCoreExtensions.cs
IdentityExtensions.cs
Migrations
Models
appsettings.json
AuthECAPI.http
Program.cs
WeatherForecast.cs
```

Angular Login & Logout with Asp.Net Core Identity & JWT

```
EFCore...ions.cs  Identity...sions.cs  Program.cs  Email
```

```
AuthECAPI
```

```
UserRegistrationModel
```

```
// Add services to the container.

builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

builder.Services.AddIdentityHandlersAndStores()
    .ConfigureIdentityOptions()
    .InjectDbContext(builder.Configuration);
```

```
(extension) IServiceCollection.ServiceCollection.InjectDbContext(Configuration config)
```

```
builder.Services.AddAuthentication(x =>
{
    x.DefaultAuthenticateScheme =
    x.DefaultChallengeScheme =
    x.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
}).AddJwtBearer(y =>
{
    y.SaveToken = false;
    y.TokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuerSigningKey = true,
        IssuerSigningKey = new SymmetricSecurityKey(
```

```
No issues found
```

```
Solution Explorer
Search Solution Explorer (Ctrl+.)
Solution 'AuthECAPI' (1 of 1 project)
AuthECAPI
Connected Services
Dependencies
Properties
Controllers
Extensions
EFCoreExtensions.cs
IdentityExtensions.cs
Migrations
Models
appsettings.json
AuthECAPI.http
Program.cs
WeatherForecast.cs
```

Angular Login & Logout with Asp.Net Core Identity & JWT

```
EFCoreExtensions.cs  IdentityExtensions.cs*  Program.cs
AuthECAPI           %AuthECAPI.Extensions.IdentityExtensions          AddIdentityAuth(IServiceCollection services)
{
    options.Password.RequireLowercase = false;
    options.User.RequireUniqueEmail = true;
}

//Auth = Authentication + Authorization
public static IServiceCollection AddIdentityAuth(this IServiceCollection services)
{
    services.AddAuthentication(x =>
    {
        x.DefaultAuthenticateScheme =
        x.DefaultChallengeScheme =
        x.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
    }).AddJwtBearer(y =>
    {
        y.SaveToken = false;
        y.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuerSigningKey = true,
            IssuerSigningKey = new SymmetricSecurityKey(
                Encoding.UTF8.GetBytes(
                    builder.Configuration["AppSettings:JWTSecret"]!))
        };
    });
    return services;
}
```

Solution Explorer

- Search Solution Explorer (Ctrl+.)
- Solution 'AuthECAPI' (1 of 1 project)
 - Solution Items
 - AuthECAPI
 - Connected Services
 - Dependencies
 - Properties
 - Controllers
 - Extensions
 - EFCoreExtensions.cs
 - IdentityExtensions.cs
 - Migrations
 - Models
 - appsettings.json
 - AuthECAPI.http
 - Program.cs
 - WeatherForecast.cs

Angular Login & Logout with Asp.Net Core Identity & JWT

```
EFCoreExtensions.cs  IdentityExtensions.cs*  Program.cs
AuthECAPI           %AuthECAPI.Extensions.IdentityExtensions          AddIdentityAuth(IServiceCollection services)
{
    options.Password.RequireDigit = false;
    options.Password.RequireUppercase = false;
    options.Password.RequireLowercase = false;
    options.User.RequireUniqueEmail = true;
};

return services;
}

//Auth = Authentication + Authorization
public static IServiceCollection AddIdentityAuth(this IServiceCollection services)
{
    services.AddAuthentication(x =>
    {
        x.DefaultAuthenticateScheme =
        x.DefaultChallengeScheme =
        x.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
    }).AddJwtBearer(y =>
    {
        y.SaveToken = false;
        y.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuerSigningKey = true,
            IssuerSigningKey = new SymmetricSecurityKey(
                Encoding.UTF8.GetBytes(
                    builder.Configuration["AppSettings:JWTSecret"]!))
        };
    });
}
```

Solution Explorer

- Search Solution Explorer (Ctrl+.)
- Solution 'AuthECAPI' (1 of 1 project)
 - Solution Items
 - AuthECAPI
 - Connected Services
 - Dependencies
 - Properties
 - Controllers
 - Extensions
 - EFCoreExtensions.cs
 - IdentityExtensions.cs
 - Migrations
 - Models
 - appsettings.json
 - AuthECAPI.http
 - Program.cs
 - WeatherForecast.cs

Angular Login & Logout with Asp.Net Core Identity & JWT

```
EFCore...ons.cs  x  Identity...ions.cs  Program.cs
AuthECAP
using AuthECAPI.Models;
using Microsoft.EntityFrameworkCore;

namespace AuthECAPI.Extensions
{
    public static class EFCoreExtensions
    {
        public static IServiceCollection InjectDbContext(
            this IServiceCollection services,
            IConfiguration config)
        {
            services.AddDbContext<AppDbContext>(options =>
                options.UseSqlServer(config.GetConnectionString("DevDB")));
            return services;
        }
    }
}
```

Solution Explorer

- Search Solution Explorer (Ctrl+.)
- Solution 'AuthECAP' (1 of 1 project)
 - AuthECAP
 - Connected Services
 - Dependencies
 - Properties
 - Controllers
 - Extensions
 - EFCoreExtensions.cs
 - IdentityExtensions.cs
 - Migrations
 - Models
 - appsettings.json
 - AuthECAP.Http
 - Program.cs
 - WeatherForecast.cs

done here

File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help Full Screen

EFCore...ons.cs Identity...ions.cs Program.cs

AuthECAP

No issues found

File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help Full Screen

EFCore...ons.cs Identity...ions.cs Program.cs

AuthECAP

No issues found

// Add services to the container.

```
builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

builder.Services.AddIdentityHandlersAndStores()
    .ConfigureIdentityOptions()
    .InjectDbContext(builder.Configuration);
```

-builder.Services.AddAuthentication(x =>

```
{ x.DefaultAuthenticateScheme =
x.DefaultChallengeScheme =
x.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
}).AddJwtBearer(y =>
{
    y.SaveToken = false;
    y.TokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuerSigningKey = true,
        IssuerSigningKey = new SymmetricSecurityKey(
```

Error List Package Manager Console Output

Ready 12:48 / 1:42:40 - Organize Program.cs File >

```
File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help Full Screen
```

EFCore...ons.cs Identity...ions.cs Program.cs

AuthECAP

No issues found

```
// Add services to the container.
```

```
builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

builder.Services.AddIdentityHandlersAndStores()
    .ConfigureIdentityOptions()
    .InjectDbContext(builder.Configuration);
```

```
-builder.Services.AddAuthentication(x =>
```

```
{ x.DefaultAuthenticateScheme =
x.DefaultChallengeScheme =
x.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
}).AddJwtBearer(y =>
{
    y.SaveToken = false;
    y.TokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuerSigningKey = true,
        IssuerSigningKey = new SymmetricSecurityKey(
```

Error List Package Manager Console Output

Ready 12:48 / 1:42:40 - Organize Program.cs File >

Angular Login & Logout with Asp.Net Core Identity & JWT

```
EFCore...ons.cs  Identity...ons.cs  Program.cs
AuthECAPI
builder.Services.AddIdentityHandlersAndStores()
    .ConfigureIdentityOptions()
    .InjectDbContext(builder.Configuration);

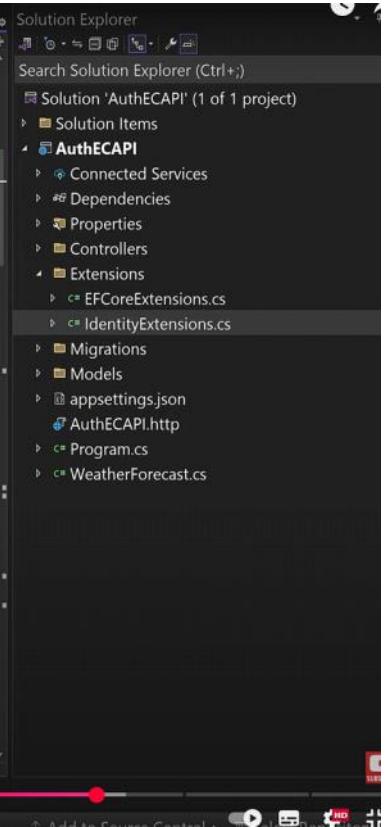
builder.Services.AddAuthentication(x =>
{
    x.DefaultAuthenticateScheme =
    x.DefaultChallengeScheme =
    x.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
}).AddJwtBearer(y =>
{
    y.SaveToken = false;
    y.TokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuerSigningKey = true,
        IssuerSigningKey = new SymmetricSecurityKey(
            Encoding.UTF8.GetBytes(
                builder.Configuration["AppSettings:JWTSecret"]!))
    };
});

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
    app.UseDeveloperExceptionPage();
else
    app.UseExceptionHandler("/Error");
    app.UseHsts();
```

No issues found

1:22:07 / 1:42:40 • Organize Program.cs File



Angular Login & Logout with Asp.Net Core Identity & JWT

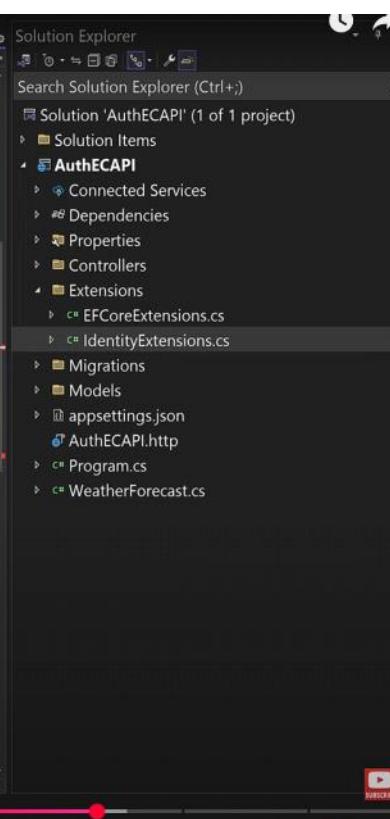
```
EFCore...ons.cs  Identity...ons.cs  Program.cs
AuthECAPI
{
    options.Password.RequireDigit = false;
    options.Password.RequireUppercase = false;
    options.Password.RequireLowercase = false;
    options.User.RequireUniqueEmail = true;
};

return services;

//Auth = Authentication + Authorization
public static IServiceCollection AddIdentityAuth(this IServiceCollection services)
{
    services.AddAuthentication(x =>
    {
        x.DefaultAuthenticateScheme =
        x.DefaultChallengeScheme =
        x.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
    }).AddJwtBearer(y =>
    {
        y.SaveToken = false;
        y.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuerSigningKey = true,
            IssuerSigningKey = new SymmetricSecurityKey(
                Encoding.UTF8.GetBytes(
                    builder.Configuration["AppSettings:JWTSecret"]!))
        };
    });
}
```

No issues found

1:33 / 1:42:40 • Organize Program.cs File



Angular Login & Logout with Asp.Net Core Identity & JWT

```
EFCoreExtensions.cs  IdentityExtensions.cs  Program.cs  AddIdentityAuth(IServiceCollection services)
```

```
AuthECAPI
```

```
{  
    options.Password.RequireDigit = false;  
    options.Password.RequireUppercase = false;  
    options.Password.RequireLowercase = false;  
    options.User.RequireUniqueEmail = true;  
};  
return services;  
  
//Auth = Authentication + Authorization  
public static IServiceCollection AddIdentityAuth(  
    this IServiceCollection services,  
    IConfiguration config)  
{  
    services.AddAuthentication(x =>  
    {  
        x.DefaultAuthenticateScheme =  
        x.DefaultChallengeScheme =  
        x.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;  
    }).AddJwtBearer(y =>  
    {  
        y.SaveToken = false;  
        y.TokenValidationParameters = new TokenValidationParameters  
        {  
            ValidateIssuerSigningKey = true,  
            IssuerSigningKey = new SymmetricSecurityKey(  
                Encoding.UTF8.GetBytes(  
                    builder.Configuration["AppSettings:JWTSecret"]!))  
        };  
    });  
}  
return services;
```

Solution Explorer

```
Search Solution Explorer (Ctrl+.)  
Solution 'AuthECAPI' (1 of 1 project)  
Solution Items  
AuthECAPI  
Connected Services  
Dependencies  
Properties  
Controllers  
Extensions  
EFCoreExtensions.cs  
IdentityExtensions.cs  
Migrations  
Models  
appsettings.json  
AuthECAPI.http  
Program.cs  
WeatherForecast.cs
```

Angular Login & Logout with Asp.Net Core Identity & JWT

```
EFCoreExtensions.cs  IdentityExtensions.cs  Program.cs  AddIdentityAuth(IServiceCollection services, ICo
```

```
AuthECAPI
```

```
{  
    options.Password.RequireLowercase = false;  
    options.User.RequireUniqueEmail = true;  
};  
return services;  
  
//Auth = Authentication + Authorization  
public static IServiceCollection AddIdentityAuth(  
    this IServiceCollection services,  
    IConfiguration config)  
{  
    services.AddAuthentication(x =>  
    {  
        x.DefaultAuthenticateScheme =  
        x.DefaultChallengeScheme =  
        x.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;  
    }).AddJwtBearer(y =>  
    {  
        y.SaveToken = false;  
        y.TokenValidationParameters = new TokenValidationParameters  
        {  
            ValidateIssuerSigningKey = true,  
            IssuerSigningKey = new SymmetricSecurityKey(  
                Encoding.UTF8.GetBytes(  
                    config["AppSettings:JWTSecret"]!))  
        };  
    });  
}  
return services;
```

Solution Explorer

```
Search Solution Explorer (Ctrl+.)  
Solution 'AuthECAPI' (1 of 1 project)  
Solution Items  
AuthECAPI  
Connected Services  
Dependencies  
Properties  
Controllers  
Extensions  
EFCoreExtensions.cs  
IdentityExtensions.cs  
Migrations  
Models  
appsettings.json  
AuthECAPI.http  
Program.cs  
WeatherForecast.cs
```

```
Angular Login & Logout with Asp.NET Core Identity & SWR
EFCore...ions.cs           Identit...sions.cs      Program.cs*  x
AuthECAP1                  * UserRegistrationModel  Email

using System.Text;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

builder.Services.AddIdentityHandlersAndStores()
    .ConfigureIdentityOptions()
    .InjectDbContext(builder.Configuration)
    .AddIdentityAuth(builder.Configuration);
```

```
// Add services to the container.

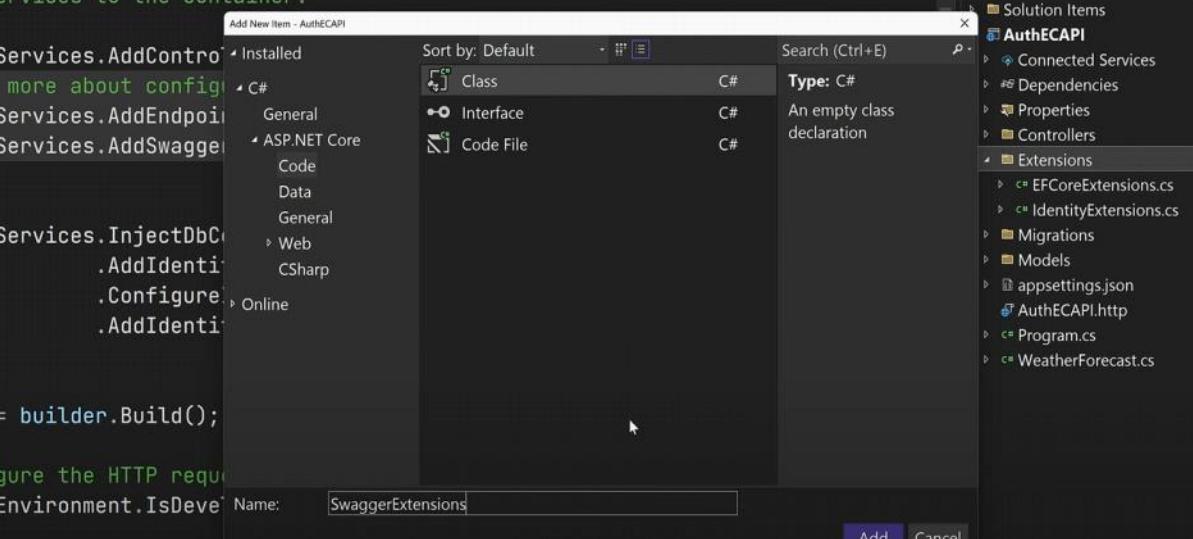
builder.Services.AddControllers();
// Learn more about configuration: https://go.microsoft.com/fwlink/?linkid=808683
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

builder.Services.InjectDbContext();
    .AddIdentity();
    .Configure();
        .AddIdentity();
        .AddIdentity();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}
```

The screenshot shows an 'Add New Item' dialog box in Visual Studio. The 'Type' dropdown is set to 'C#' and the 'Class' option is selected. The 'Name:' field contains 'SwaggerExtensions'. The 'Add' and 'Cancel' buttons are visible at the bottom right of the dialog.



The screenshot shows the Visual Studio IDE interface. The Solution Explorer on the right lists a single project named 'AuthECAPI' containing files like Program.cs, EFCoreExtensions.cs, IdentityExtensions.cs, SwaggerExtensions.cs, Migrations, Models, appsettings.json, AuthECAPI.http, and WeatherForecast.cs. The code editor on the left displays the 'SwaggerExtensions.cs' file, which contains a static class 'SwaggerExtensions' with a method 'AddSwaggerExplorer' that adds identity endpoints and entity framework stores to an IServiceCollection.

```
Swagg...ns.cs* x Program.cs EFCore...ions.cs Identit...sions.cs
AuthECAPI AddSwaggerExplorer(IServiceCollection services)
using AuthECAPI.Models;

namespace AuthECAPI.Extensions
{
    public static class SwaggerExtensions
    {
        public static IServiceCollection AddSwaggerExplorer(this IServiceCollection services)
        {
            services.AddIdentityApiEndpoints<AppUser>()
                .AddEntityFrameworkStores<AppDbContext>();
            return services;
        }
    }
}
```

The screenshot shows the Visual Studio IDE interface with the 'Program.cs' file open. The code defines a WebApplication builder, adds controllers and the Swagger Explorer, injects a database context, and configures identity options and authentication. It then builds the application and configures the HTTP request pipeline by enabling Swagger and its UI if the environment is development.

```
wagge...ons.cs Program.cs* x EFCore...ions.cs Identit...sions.cs
AuthECAPI UserRegistrationModel Email
using System.Text;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllers();
builder.Services.AddSwaggerExplorer()
    .InjectDbContext(builder.Configuration)
    .AddIdentityHandlersAndStores()
    .ConfigureIdentityOptions()
    .AddIdentityAuth(builder.Configuration);

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}
```

Angular Login & Logout with Asp.Net Core Identity & JWT

Program.cs x Swagg...ns.cs* EFCore...lions.cs Identit...sions.cs

AuthECAPI UserRegistrationModel Email

```
// Add services to the container.
builder.Services.AddControllers();
builder.Services.AddSwaggerExplorer()
    .InjectDbContext(builder.Configuration)
    .AddIdentityHandlersAndStores()
    .ConfigureIdentityOptions()
    .AddIdentityAuth(builder.Configuration);

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

Config.CORS
app.UseAuthentication();
app.UseAuthorization();

app.MapControllers();

app
    .MapGroup("/api")
    .MapIdentityApi<AppUser>();
```

No issues found | 1:25:32 / 1:42:40 • Organize Program.cs File >

Solution Explorer

Search Solution Explorer (Ctrl+.)

Solution 'AuthECAPI' (1 of 1 project)

AuthECAPI

- Connected Services
- Dependencies
- Properties
- Controllers
- Extensions
 - EFCoreExtensions.cs
 - IdentityExtensions.cs
 - SwaggerExtensions.cs
- Migrations
- Models
- appsettings.json
- AuthECAPI.http
- Program.cs
- WeatherForecast.cs

Angular Login & Logout with Asp.Net Core Identity & JWT

Program.cs* x Swagg...ns.cs* EFCore...lions.cs Identit...sions.cs

AuthECAPI AuthECAPI.Extensions.SwaggerExtensions ConfigureSwaggerExplorer(WebApplication app)

```
using AuthECAPI.Models;

namespace AuthECAPI.Extensions
{
    public static class SwaggerExtensions
    {
        public static IServiceCollection AddSwaggerExplorer(this IServiceCollection services)
        {
            // Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
            services.AddEndpointsApiExplorer();
            services.AddSwaggerGen();
            return services;
        }

        public static WebApplication ConfigureSwaggerExplorer(this WebApplication app)
        {
            // Configure the HTTP request pipeline.
            if (app.Environment.IsDevelopment())
            {
                app.UseSwagger();
                app.UseSwaggerUI();
            }
            return app;
        }
    }
}
```

No issues found | 1:25:36 / 1:42:40 • Organize Program.cs File >

Solution Explorer

Search Solution Explorer (Ctrl+.)

Solution 'AuthECAPI' (1 of 1 project)

AuthECAPI

- Connected Services
- Dependencies
- Properties
- Controllers
- Extensions
 - EFCoreExtensions.cs
 - IdentityExtensions.cs
 - SwaggerExtensions.cs
- Migrations
- Models
- appsettings.json
- AuthECAPI.http
- Program.cs
- WeatherForecast.cs

Angular Login & Logout with Asp.Net Core Identity & JWT

```
Program.cs*  x Swagger...ons.cs  EFCore...ions.cs  Identit...ions.cs
AuthECAPI  UserRegistrationModel  Email
```

```
// Add services to the container.
builder.Services.AddControllers();
builder.Services.AddSwaggerExplorer()
    .InjectDbContext(builder.Configuration)
    .AddIdentityHandlersAndStores()
    .ConfigureIdentityOptions()
    .AddIdentityAuth(builder.Configuration);

var app = builder.Build();

app.ConfigureSwaggerExplorer

#Config. CORS
app.UseAuthentication();
app.UseAuthorization();

app.MapControllers();

app
    .MapGroup("/api")
    .MapIdentityApi<AppUser>();

app.MapPost("/api/signup", async (
    UserManager<AppUser> userManager,
    [FromBody] UserRegistrationModel userRegistrationModel
)
```

1:25:41 / 1:42:40 - Organize Program.cs File >

Solution Explorer

- Solution 'AuthECAPI' (1 of 1 project)
 - Solution Items
 - AuthECAPI
 - Connected Services
 - Dependencies
 - Properties
 - Controllers
 - Extensions
 - EFCoreExtensions.cs
 - IdentityExtensions.cs
 - SwaggerExtensions.cs
 - Migrations
 - Models
 - appsettings.json
 - AuthECAPI.http
 - Program.cs
 - WeatherForecast.cs

Angular Login & Logout with Asp.Net Core Identity & JWT

```
Swagge...ons.cs  Identit...ions.cs  EFCore...ions.cs  AppCon...ons.cs  Program.cs*  Weather...ller.cs  202407...ity2.cs
AuthECAPI  UserRegistrationModel  Email
```

```
var app = builder.Build();

app.ConfigureSwaggerExpl

#region Config. CORS
app.UseCors(options =>
    options.WithOrigins(
        .AllowAnyMet
        .AllowAnyHea
    )
);
app.UseAuthentication();
app.UseAuthorization();

app.MapControllers();

app
    .MapGroup("/api")
    .MapIdentityApi<AppUser>();
```

Add New Item - AuthECAPI

Sort by: Name Ascendin	Class	C#
	Code File	C#
	Interface	C#

Type: C#
An empty class declaration

Name: AppConfigExtensions

Add Cancel

Solution Explorer

- Solution 'AuthECAPI' (1 of 1 project)
 - Solution Items
 - AuthECAPI
 - Connected Services
 - Dependencies
 - Properties
 - Controllers
 - Extensions
 - EFCoreExtensions.cs
 - IdentityExtensions.cs
 - SwaggerExtensions.cs
 - Migrations
 - Models
 - appsettings.json
 - AuthECAPI.http
 - Program.cs
 - WeatherForecast.cs

The screenshot shows the Visual Studio IDE interface with the 'AuthECAPI' project open. The Solution Explorer on the right lists files like 'AuthECAPI.csproj', 'Connected Services', 'Dependencies', 'Properties', 'Controllers', and 'Extensions'. The 'Extensions' folder contains files such as 'AppConfigExtensions.cs', 'EFCoreExtensions.cs', 'IdentityExtensions.cs', 'SwaggerExtensions.cs', 'Migrations', 'Models', 'appsettings.json', 'AuthECAPI.http', 'Program.cs', and 'WeatherForecast.cs'. The main code editor window displays 'AppCo...ns.cs' with the following code:

```
AppCo...ns.cs*  Program.cs*  Swagger...ons.cs  EFCore...ions.cs  Identity...ions.cs
AuthECAPI
AuthECAPI.Extensions.AppConfigExtensions  ConfigureCORS(WebApplication app)
namespace AuthECAPI.Extensions
{
    public static class AppConfigExtensions
    {
        public static WebApplication ConfigureCORS(this WebApplication app)
        {
            // Configure the HTTP request pipeline.
            if (app.Environment.IsDevelopment())
            {
                app.UseSwagger();
                app.UseSwaggerUI();
            }
            return app;
        }
    }
}
```

This screenshot is identical to the one above, showing the 'AuthECAPI' project structure and the 'AppCo...ns.cs' file content. The code in 'AppCo...ns.cs' is as follows:

```
Swagge...ons.cs  Identit...ions.cs  EFCore...ions.cs  AppCo...ns.cs*  Program.cs*  Weather...ller.cs  202407...ity2.cs
AuthECAPI
AuthECAPI.Extensions.AppConfigExtensions  ConfigureCORS(WebApplication app)
using AuthECAPI.Models;
using Microsoft.EntityFrameworkCore;

namespace AuthECAPI.Extensions
{
    public static class AppConfigExtensions
    {
        public static WebApplication ConfigureCORS(this WebApplication app)
        {
            app.UseCors(options =>
            options.WithOrigins("http://localhost:4200")
                .AllowAnyMethod()
                .AllowAnyHeader());
            return app;
        }
    }
}
```

configureCORS

The screenshot shows the Visual Studio IDE interface with the following details:

- Solution Explorer:** Shows the project structure for "AuthECAPI".
- Program.cs:** The code is being edited.
- Code Snippet:**

```
builder.Services.AddSwaggerExplorer()
    .InjectDbContext(builder.Configuration)
    .AddIdentityHandlersAndStores()
    .ConfigureIdentityOptions()
    .AddIdentityAuth(builder.Configuration);

var app = builder.Build();

app.ConfigureSwaggerExplorer();

app.ConfigureCORS();
app.UseAuthentication();
app.UseAuthorization();

app.MapControllers();

app
    .MapGroup("/api")
    .MapIdentityApi<AppUser>();

app.MapPost("/api/signup", async (
    UserManager<AppUser> userManager,
    [FromBody] UserRegistrationModel userRegistrationModel
```
- Status Bar:** Shows "No issues found".
- Bottom Bar:** Shows "Ready", "1:26:34 / 1:42:40", "Organize Program.cs File", and "Add to Source Control".

The screenshot shows the Visual Studio IDE interface with the following details:

- Solution Explorer:** Shows the project structure for "AuthECAPI".
- Program.cs:** The code is being edited.
- Code Snippet:**

```
builder.Services.AddSwaggerExplorer()
    .InjectDbContext(builder.Configuration)
    .AddIdentityHandlersAndStores()
    .ConfigureIdentityOptions()
    .AddIdentityAuth(builder.Configuration);

var app = builder.Build();

app.ConfigureSwaggerExplorer();

app.ConfigureCORS(builder.Configuration);
app.UseAuthentication();
app.UseAuthorization();

app.MapControllers();

app
    .MapGroup("/api")
    .MapIdentityApi<AppUser>();

app.MapPost("/api/signup", async (
    UserManager<AppUser> userManager,
    [FromBody] UserRegistrationModel userRegistrationModel
```
- Status Bar:** Shows "No issues found".
- Bottom Bar:** Shows "Ready", "1:26:48 / 1:42:40", "Organize Program.cs File", and "Add to Source Control".

```
namespace AuthECAPI.Extensions
{
    public static class AppConfigExtensions
    {
        public static WebApplication ConfigureCORS(
            this WebApplication app,
            IConfiguration config)
        {
            app.UseCors(options =>
            options.WithOrigins("http://localhost:4200")
                .AllowAnyMethod()
                .AllowAnyHeader());
            return app;
        }
    }
}
```

Output window:

```
119% No issues found | 1:27:01 / 1:42:40 - Organize Program.cs File >
```

```
services.AddEndpointsApiExplorer();
services.AddSwaggerGen();
return services;

public static WebApplication ConfigureSwaggerExplorer(this WebApplication app)
{
    // Configure the HTTP request pipeline.
    if (app.Environment.IsDevelopment())
    {
        app.UseSwagger();
        app.UseSwaggerUI();
    }
    return app;
}
```

Output window:

```
119% No issues found | 1:27:01 / 1:42:40 - Organize Program.cs File >
```

The screenshot shows the Visual Studio IDE interface with the following details:

- File Bar:** File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help.
- Solution Explorer:** Shows the solution 'AuthEC API' with one project item: 'AuthEC API'.
- Code Editor:** The active file is 'IdentityExtensions.cs' under the 'Extensions' folder. The code implements the `IIdentityBuilder` interface with methods to configure password requirements and add identity authentication and authorization middlewares.
- Status Bar:** Shows 'No issues found' and other standard status indicators.

```
options.Password.RequireDigit = false;
options.Password.RequireUppercase = false;
options.Password.RequireLowercase = false;
options.User.RequireUniqueEmail = true;

});

return services;

}

//Auth = Authentication + Authorization
public static IServiceCollection AddIdentityAuth(
    this IServiceCollection services,
    IConfiguration config)...

public static WebApplication AddIdentityAuthMiddlewares(this WebApplication app)
{
    return app;
}
}
```

The screenshot shows the Visual Studio IDE interface with the following details:

- File Bar:** File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help.
- Solution Explorer:** Shows the solution 'AuthEC API' with one project item: 'AuthEC API'.
- Code Editor:** The active file is 'IdentityExtensions.cs' under the 'Extensions' folder. The cursor is positioned over the `AddIdentityAuthMiddlewares` method.
- Status Bar:** Shows 'No issues found' and other standard status indicators.

```
options.Password.RequireDigit = false;
options.Password.RequireUppercase = false;
options.Password.RequireLowercase = false;
options.User.RequireUniqueEmail = true;

});

return services;

}

//Auth = Authentication + Authorization
public static IServiceCollection AddIdentityAuth(
    this IServiceCollection services,
    IConfiguration config)...

public static WebApplication AddIdentityAuthMiddlewares(this WebApplication app)
{
    app.UseAuthentication();
    app.UseAuthorization();
    return app;
}
}
```

```
builder.Services.AddSwaggerExplorer()
    .InjectDbContext(builder.Configuration)
    .AddIdentityHandlersAndStores()
    .ConfigureIdentityOptions()
    .AddIdentityAuth(builder.Configuration);

var app = builder.Build();

app.ConfigureSwaggerExplorer();

app.ConfigureCORS(builder.Configuration)
    .AddIdentityAuthMiddlewares(); // tooltip: [extension] WebApplication.WebApplication.AddIdentityAuthMiddlewares()

app.MapControllers();

app
    .MapGroup("/api")
    .MapIdentityApi<AppUser>();

app.MapPost("/api/signup", async (
    UserManager<AppUser> userManager,
    [FromBody] UserRegistrationModel userRegistrationModel
) => ...);

app.MapPost("/api/signin", async (
    UserManager<AppUser> userManager,
    [FromBody] LoginModel loginModel
) => ...);

app.Run();

public class UserRegistrationModel
{
    public string Email { get; set; }
    public string Password { get; set; }
    public string FullName { get; set; }
}
```

No issues found

1:27:52 / 1:42:40 • Organize Program.cs File >

```
app.MapControllers();
app.MapGroup("/api")
    .MapIdentityApi<AppUser>();

app.MapPost("/api/signup", async (
    UserManager<AppUser> userManager,
    [FromBody] UserRegistrationModel userRegistrationModel
) => ...);

app.MapPost("/api/signin", async (
    UserManager<AppUser> userManager,
    [FromBody] LoginModel loginModel
) => ...);

app.Run();

public class UserRegistrationModel
{
    public string Email { get; set; }
    public string Password { get; set; }
    public string FullName { get; set; }
}
```

Add New Item - AuthECAPI

Sort by: Default

Type: C#

Name: IdentityUserEndpoints

Add Cancel

No issues found

1:29:47 / 1:42:40 • Organize Web API Endpoints >

Mapidentityuserendpoing

```
app.ConfigureSwaggerExplorer()
    .ConfigureCORS(builder.Configuration)
    .AddIdentityAuthMiddlewares();

app.MapControllers();
app.MapGroup("/api")
    .MapIdentityApi<AppUser>();
app.MapIdentityUserEndpoints();

app.MapPost("/api/signup", async (
    UserManager<AppUser> userManager,
    [FromBody] UserRegistrationModel userRegistrationModel
) => ...);

app.MapPost("/api/signin", async (
    UserManager<AppUser> userManager,
    [FromBody] LoginModel loginModel) => ...);

app.Run();

public class UserRegistrationModel
```

```
.AddIdentityAuthMiddlewares();

app.MapControllers();
app.MapGroup("/api")
    .MapIdentityApi<AppUser>();
app.MapIdentityUserEndpoints();

app.MapPost("/api/signup", async (
    UserManager<AppUser> userManager,
    [FromBody] UserRegistrationModel userRegistrationModel
) => ...);

app.MapPost("/api/signin", async (
    UserManager<AppUser> userManager,
    [FromBody] LoginModel loginModel) => ...);

app.Run();

public class UserRegistrationModel
```

Angular Login & Logout with Asp.Net Core Identity & JWT

```
Ident...nts.cs Weather...ller.cs AppCon...ons.cs Program.cs* EFCore...ions.cs Ident...sions.cs
AuthECAPI
namespace AuthECAPI.Controllers
{
    public static class IdentityUserEndpoints
    {
        public static IEndpointRouteBuilder MapIdentityUserEndpoints(this IEndpointRouteBuilder app)
        {
            return app;
        }
    }
}
```

Solution Explorer

- Solution 'AuthECAPI' (1 of 1 project)
 - Solution Items
 - AuthECAPI
 - Connected Services
 - Dependencies
 - Properties
 - Controllers
 - IdentityUserEndpoints.cs
 - WeatherForecastController.cs
 - Extensions
 - AppConfigExtensions.cs
 - EFCoreExtensions.cs
 - IdentityExtensions.cs
 - SwaggerExtensions.cs
 - Migrations
 - Models
 - appsettings.json
 - AuthECAPI.http
 - Program.cs
 - WeatherForecast.cs

```
Ident...oints.cs Weather...ller.cs AppCon...ons.cs Program.cs* EFCore...ions.cs Ident...sions.cs
AuthECAPI
app.MapIdentityUserEndpoints();

app.Run();

public class UserRegistrationModel
{
    public string Email { get; set; }
    public string Password { get; set; }
    public string FullName { get; set; }
}

public class LoginModel
{
    public string Email { get; set; }
    public string Password { get; set; }
}
```

Solution Explorer

- Solution 'AuthECAPI' (1 of 1 project)
 - Solution Items
 - AuthECAPI
 - Connected Services
 - Dependencies
 - Properties
 - Controllers
 - IdentityUserEndpoints.cs
 - WeatherForecastController.cs
 - Extensions
 - AppConfigExtensions.cs
 - EFCoreExtensions.cs
 - IdentityExtensions.cs
 - SwaggerExtensions.cs
 - Migrations
 - Models
 - appsettings.json
 - AuthECAPI.http
 - Program.cs
 - WeatherForecast.cs

GitHub - CodAffection/Auth - Angular-JWT-Login-Logout-with-.Net-Core-API-Asp.Net-Core-Identity

Angular-JWT-Login-Logout-with-.Net-Core-API-Asp.Net-Core-Identity / AuthECAPI / AuthECAPI / Controllers / IdentityUserEndpoints.cs

Code Blame 86 lines (80 loc) · 3.18 KB

```
1  using AuthECAPI.Models;
2  using Microsoft.AspNetCore.Identity;
3  using Microsoft.AspNetCore.Mvc;
4  using Microsoft.Extensions.Options;
5  using Microsoft.IdentityModel.Tokens;
6  using System.IdentityModel.Tokens.Jwt;
7  using System.Security.Claims;
8  using System.Text;
9
10 namespace AuthECAPI.Controllers
11 {
12     public class UserRegistrationModel
13     {
14         public string Email { get; set; }
15         public string Password { get; set; }
16         public string FullName { get; set; }
17     }
18
19     public class LoginModel
20     {
21         public string Email { get; set; }
22         public string Password { get; set; }
23     }
24
25     public static class IdentityUserEndpoints
26     {
27         public static IEndpointRouteBuilder MapIdentityUserEndpoints(this IEndpointRouteBuilder app)
28         {
29             app.MapPost("/signup", CreateUser);
30         }
31     }
32 }
```

Symbols

Find definitions and references for functions and other symbols in this file by clicking a symbol below or in the code.

Filter symbols

- class UserRegistrationModel
- class LoginModel
- class IdentityUserEndpoints
 - func MapIdentityUserEndpoints
 - func CreateUser
 - func SignIn

GitHub - CodAffection/Auth - Angular-JWT-Login-Logout-with-.Net-Core-API-Asp.Net-Core-Identity

Angular-JWT-Login-Logout-with-.Net-Core-API-Asp.Net-Core-Identity / AuthECAPI / AuthECAPI / Controllers / IdentityUserEndpoints.cs

Code Blame 86 lines (80 loc) · 3.18 KB

```
18
19     public class LoginModel
20     {
21         public string Email { get; set; }
22         public string Password { get; set; }
23     }
24
25     public static class IdentityUserEndpoints
26     {
27         public static IEndpointRouteBuilder MapIdentityUserEndpoints(this IEndpointRouteBuilder app)
28         {
29             app.MapPost("/signup", CreateUser);
30             app.MapPost("/signin", SignIn);
31             return app;
32         }
33     }
34     private static async Task<IResult> CreateUser(
35         UserManager<AppUser> userManager,
36         [FromBody] UserRegistrationModel userRegistrationModel)
37     {
38         AppUser user = new AppUser()
39         {
40             UserName = userRegistrationModel.Email,
41             Email = userRegistrationModel.Email,
42             FullName = userRegistrationModel.FullName,
43         };
44         var result = await userManager.CreateAsync(
45             user,
46             userRegistrationModel.Password);
47
48         if (result.Succeeded)
49             return Results.Ok(result);
50     }
51 }
```

Symbols

Find definitions and references for functions and other symbols in this file by clicking a symbol below or in the code.

Filter symbols

- class UserRegistrationModel
- class LoginModel
- class IdentityUserEndpoints
 - func MapIdentityUserEndpoints
 - func CreateUser
 - func SignIn

GitHub - CodAffection/Angular-JWT-Login-Logout-with-.Net-Core-API-Asp.Net-Core-Identity

Angular-JWT-Login-Logout-with-.Net-Core-API-Asp.Net-Core-Identity / AuthECAPI / AuthECAPI / Controllers / IdentityUserEndpoints.cs

Code Blame 86 lines (80 loc) · 3.18 KB

```
25     public static class IdentityUserEndpoints
26     {
27         [HttpGet]
28         public IActionResult SignIn([FromBody] LoginModel loginModel)
29         {
30             var user = await userManager.FindByEmailAsync(loginModel.Email);
31             if (user != null && await userManager.CheckPasswordAsync(user, loginModel.Password))
32             {
33                 var signInKey = new SymmetricSecurityKey(
34                     Encoding.UTF8.GetBytes(appSettings.Value.JWTSecret));
35                 var tokenDescriptor = new SecurityTokenDescriptor
36                 {
37                     Subject = new ClaimsIdentity(new Claim[]
38                     {
39                         new Claim("UserID", user.Id.ToString())
40                     }),
41                     Expires = DateTime.UtcNow.AddDays(10),
42                     SigningCredentials = new SigningCredentials(
43                         signInKey,
44                         SecurityAlgorithms.HmacSha256Signature
45                     );
46                 };
47                 var tokenHandler = new JwtSecurityTokenHandler();
48                 var securityToken = tokenHandler.CreateToken(tokenDescriptor);
49                 var token = tokenHandler.WriteToken(securityToken);
50                 return Ok(new { token });
51             }
52         }
53     }
```

Symbols

Find definitions and references for functions and other symbols in this file by clicking a symbol below or in the code.

Filter symbols

class UserRegistrationModel

class LoginModel

class IdentityUserEndpoints

func MapIdentityUserEndpoints

func CreateUser

func SignIn

GitHub - CodAffection/Angular-JWT-Login-Logout-with-.Net-Core-API-Asp.Net-Core-Identity

Angular-JWT-Login-Logout-with-.Net-Core-API-Asp.Net-Core-Identity / AuthECAPI / AuthECAPI / Program.cs

Code Blame 39 lines (30 loc) · 1.04 KB

```
6     using Microsoft.AspNetCore.Mvc;
7     using Microsoft.EntityFrameworkCore;
8     using Microsoft.IdentityModel.Tokens;
9     using System.IdentityModel.Tokens.Jwt;
10    using System.Security.Claims;
11    using System.Text;
12
13    var builder = WebApplication.CreateBuilder(args);
14
15    // Add services to the container.
16    builder.Services.AddControllers();
17    builder.Services.AddSwaggerExplorer()
18        .InjectDbContext(builder.Configuration)
19        .AddAppConfig(builder.Configuration)
20        .AddIdentityHandlersAndStores()
21        .ConfigureIdentityOptions()
22        .AddIdentityAuth(builder.Configuration);
23
24    var app = builder.Build();
25
26    app.ConfigureSwaggerExplorer()
27        .ConfigureCORS(builder.Configuration)
28        .AddIdentityAuthMiddlewares();
29
30    app.MapControllers();
31    app.MapGroup("/api")
32        .MapIdentityApi<AppUser>();
33    app.MapGroup("/api")
34        .MapIdentityUserEndpoints();
35
36    app.Run();
```

The screenshot shows a Visual Studio IDE window with the following details:

- Title Bar:** Angular Login & Logout with Asp.Net Core Identity & SWI
- Solution Explorer:** Shows files like IdentityUserEndpoints.cs, OrderEndpoints.cs, WeatherForecastController.cs, etc.
- Code Editor:** Displays a portion of the AuthECAPI.cs file containing code for user authentication and identity management.
- Add New Item Dialog:** A modal window titled "Add New Item - AuthECAPI" is open, showing the "C#" category under "Type: C#".
 - Class:** Selected item.
 - Interface:**
 - Code File:**
- Search Bar:** Shows "Type: C#".
- Bottom Status Bar:** Package Manager Console, Output, and other development-related information.

The screenshot shows a GitHub repository page for 'AuthECAPI' under the 'AuthECAPI' organization. The file 'AppSettings.cs' is selected in the code editor. The code defines a class 'AppSettings' with a single public string property 'JWTSecret'. The GitHub interface includes a sidebar with project files like 'AppConfigExtensions.cs', 'EFCoreExtensions.cs', 'IdentityExtensions.cs', 'SwaggerExtensions.cs', 'Migrations', 'Models' (containing 'AppDbContext.cs', 'AppSettings.cs', 'AppUser.cs'), 'Properties', 'bin/Debug/net8.0', 'obj', 'AuthECAPI.csproj', 'AuthECAPI.http', 'Program.cs', 'WeatherForecast.cs', 'appsettings.Development.json', 'appsettings.json', and '.editorconfig'. The commit history shows 'first commit' by 'CodAffection' at 'a9b57ae' 6 months ago.

```

1  namespace AuthECAPI.Models
2  {
3      public class AppSettings
4      {
5          public string JWTSecret { get; set; }
6      }
7  }

```

The screenshot shows the 'Program.cs' file in Visual Studio. The code uses the Microsoft.Extensions.DependencyInjection builder pattern to configure services. It adds identity handlers, configures identity options, adds identity auth, and maps controllers. It also configures Swagger Explorer, CORS, and Identity auth middlewares. Finally, it runs the application. The Solution Explorer on the right shows the project structure with files like 'AuthECAPI.csproj', 'Connected Services', 'Dependencies', 'Properties', 'Controllers' (containing 'IdentityUserEndpoints.cs', 'OrderEndpoints.cs', 'WeatherForecastController.cs'), 'Extensions' (containing 'AppConfigExtensions.cs', 'EFCoreExtensions.cs', 'IdentityExtensions.cs', 'SwaggerExtensions.cs'), 'Migrations', 'Models' (containing 'AppDbContext.cs', 'AppSettings.cs', 'AppUser.cs'), 'appsettings.json', 'AuthECAPI.http', 'Program.cs', and 'WeatherForecast.cs'.

```

AuthECAPI
    .AddIdentityHandlersAndStores()
    .ConfigureIdentityOptions()
    .AddIdentityAuth(builder.Configuration);

builder.Services.Configure<AppSettings>(builder.Configuration.GetSection("AppSettings"));

var app = builder.Build();

app.ConfigureSwaggerExplorer()
    .ConfigureCORS(builder.Configuration)
    .AddIdentityAuthMiddlewares();

app.MapControllers();
app.MapGroup("/api")
    .MapIdentityApi<AppUser>();
app.MapGroup("/api")
    .MapIdentityUserEndpoints();

app.Run();

```

The screenshot shows a Visual Studio interface with the following details:

- Solution Explorer:** Shows the project structure for 'AuthECAPI'. The 'Extensions' folder contains several files: AppConfigExtensions.cs, EFCoreExtensions.cs, IdentityExtensions.cs, and SwaggerExtensions.cs.
- Code Editor:** Displays the content of 'AppConfigExtensions.cs'. The code defines two static methods: 'ConfigureCORS' and 'AddAppConfig'. The 'AddAppConfig' method uses 'services.Configure<AppSettings>' to read settings from 'AppSettings.json'.
- Status Bar:** Shows the current time as 1:41:01 / 1:42:40, the taskbar position as 16, and the character count as 54.

The screenshot shows the Visual Studio IDE interface. The top menu bar includes 'File', 'Edit', 'View', 'Project', 'Build', 'Debug', 'Any CPU', 'http', 'Tools', 'Help', and 'GitHub Copilot'. The title bar displays 'Angular Login Logout with Asp.Net Core Identity Page 62'. The left side shows several tabs: 'OrderEn...nts.cs', 'appset...gs.json', 'Identit...oints.cs', 'Program.cs' (which is the active tab), 'AppCon...ons.cs', and 'EFCore...ions.cs'. The right side features the 'Solution Explorer' window, which lists the project 'AuthECAPI' (1 of 1 project) containing files like 'Connected Services', 'Dependencies', 'Properties', 'Controllers' (with 'IdentityUserEndpoints.cs', 'OrderEndpoints.cs', 'WeatherForecastController.cs'), 'Extensions' (with 'AppConfigExtensions.cs', 'EFCoreExtensions.cs', 'IdentityExtensions.cs', 'SwaggerExtensions.cs'), 'Migrations', 'Models' (with 'AppDbContext.cs', 'AppSettings.cs', 'AppUser.cs'), 'appsettings.json', 'AuthECAPI.http', 'Program.cs', and 'WeatherForecast.cs'. The status bar at the bottom shows 'In 16 Ch 35 SPC CRLF'.

```
using System.Security.Claims;
using System.Text;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllers();
builder.Services.AddSwaggerExplorer()
    .InjectDbContext(builder.Configuration)
    .AddAppConfig(builder.Configuration)
    .AddIdentityHandlersAndStores()
    .ConfigureIdentityOptions()
    .AddIdentityAuth(builder.Configuration);

var app = builder.Build();

app.ConfigureSwaggerExplorer()
    .ConfigureCORS(builder.Configuration)
    .AddIdentityAuthMiddlewares();

app.MapControllers();
app.MapGroup("/api")
    .MapIdentityApi<AppUser>();
app.MapGroup("/api")
    .MapIdentityUserEndpoints();

app.Run();

app.Run();
```