بر اساس مستند <https://learn.microsoft.com/en-us/ef/core/miscellaneous/multitenancy>  
در این مستند میزان پشتیبانی ef core از multitenancy طرح شده است. سه رویکرد برای پیاده سازس multitenancy در نظر گرفت است:  
Multiple schemas : توسط ef core به صورت رسمی پشتبیانی نمی شود . اما در صورت عدم استفاده از migration خود ef core می توان از آن بهره برد.  
single database : یک دیتابیس برای همه در نظر گرفته شود و داده ها از طریق یک ستون که آی دی Tenant را نگه می دارند تفکیک می شوند . این طرح توسط ef core پشتیبان می شود و برای پیاده سازی آن در ef core می شود از query filter استفاده کرد.  
Multiple database : این حالت نیز توسط ef core پشتیبانی می شود و می توان در فایل dbcontext با تنظیمات لازم بین connection stringهای هر tenant سوییچ کرد و ...  
در این مستند نمونه کد ساده پیاده سازی برای دو حالت اخیر در .net و با استفاده از ef core معرفی شده که از طریق لینک github ارائه شده در متن مستند قابل دستیابی است.  
<https://github.com/dotnet/EntityFramework.Docs/tree/main/samples/core/Miscellaneous/Multitenancy>

مستند <https://code-maze.com/aspnetcore-multitenant-application/>  
در این مستند پیاده سازی یک روش هیبرید که ترکیب روش یک دیتابیس برای مشتریان با حجم بار کم و همزمان دیتابیس جدا برای مشتریان با حجم بار زیاد است.  
این مستند کد پیاده سازی را در github به اشتراک گذاشته است:  
<https://github.com/CodeMazeBlog/CodeMazeGuides/tree/main/aspnetcore-webapi/MultiTenantApplication>  
در این پیاده سازی tenant در claim های استخراجی از توکن وجود دارد که توسط یک Middleware در pipline درخواست استخراج و استفاده می شود.

مستند <https://codewithmukesh.com/blog/multitenancy-in-aspnet-core/>  
در این روش نخست چهار رویکرد رایج پیاده سازی multitenancy را به صورت مختصر بررسی کرده :  
Schema Seperation  
Single Database – Tenant Column Seperation  
Multiple Database – Complete Data Isolation  
Hybrid Approach – Favorite!  
و در نهایت با توجه به کارآمدی روش هیبرید آن را پیاده سازی کرده است. برای تشخیص tenant چهار راه معرفی کرده است :  
Query String  
Request IP Address  
Request Header  
Claims  
که در پیاده سازی خود از header های request استفاده کرده است.  
همچنین سورس پیاده سازی را روی github به اشتراک گذاشته است.  
<https://github.com/iammukeshm/multitenancy-aspnet-core>  
نگارنده مقاله در پروژه زیر، محدودیت identity یوزر را در تعیین tenant نیز لحاظ کرده:  
<https://github.com/fullstackhero/dotnet-webapi-boilerplate>

روش هیبرید (وجود همزمان دیتابیس مشترک برای مشتریان کم مصرف و دیتابیس های جدا برای مشتریان پرمصرف یا حساس به جداسازی)

با توجه به امکانات Ef Core، از جهت پیاده سازی نیز با سهولت قابل توجهی روبرو هستیم.  
برای مشتریان با دیتابیس مشترک جداسازی با استفاده از ستون TenantId برای تفکیک داده ها در سطح دیتابیس و استفاده از ویژگی QueryFilter در EfCore در سطح اپلیکیشن برای جداسازی از سطح DbContext انجام می شود. لازم است که کلاس پایه entityهای دیتابیس property با عنوان TenantID را داشته باشد و entityهای دیگر از آن ارث بری کنند.  
برای مشتریان با دیتابیس جدا این تفیکیک در سطح اپلیکیشن و هنگام ساختن نمونه کلاس DbContext با استفاده از تزریق Connection string های مخصوص به دیتابیس مربوطه انجام می شود. مقاله سایت CodeMaze یک نمونه خوب از پیاده سازی این روش است که متن آموزش در لینک:  
<https://code-maze.com/aspnetcore-multitenant-application/>  
و سورس کد آن در :  
<https://github.com/CodeMazeBlog/CodeMazeGuides/tree/main/aspnetcore-webapi/MultiTenantApplication>  
موجود است.  
همچنین مستندات سایت ماکروسافت در ارتباط با قابلیت های مرتبط با EfCore در این حوزه نیز راهگشاست:  
<https://learn.microsoft.com/en-us/ef/core/miscellaneous/multitenancy>  
سورس آن :  
<https://github.com/dotnet/EntityFramework.Docs/tree/main/samples/core/Miscellaneous/Multitenancy>  
  
توجه: Efcore قابلیت Migration در حالت هیبرید را هم دارد اما با توجه به تصمیم گیری های انجام شده، از این قابلیت در این پروژه استفاده نمی شود.  
==============================  
مورد دیگر جهت تصمیم گیری انتخاب روش تشخیص tenant از درخواست مشتری است. چهار روش برای این کار ذکر شده است:  
Query String  
Request IP Address  
Request Header  
Claims  
روش Query String به دلیل امنیت پایین روش مطلوبی نیست  
روش Request IP Address امن ترین روش است اما سربار عملیاتی بالایی دارد.