📦 RiRaDocs Teaching Edition – Release Meta File

نسخه: `-RiraDocs-v2025.11.4-Stable` **`Final-Fixed`**

این سند نسخه‌ی نهایی یادداشت انتشار (Release Note) پروژه‌ی RiRa (Teaching Edition) است و شامل مستندسازی، تغییرات، و جزئیات مربوط به بخش‌های مختلف Domain، Application ،Persistence، و تست‌ها می‌باشد.

خلاصه‌ی جامع توسعه و مستندسازی

این نسخه تمرکز ویژه‌ای بر روی تثبیت لایه‌های دامنه‌ای، بهبود سازگاری با EF Core 8، و تقویت زیرساخت تست‌ها، به‌ویژه شبیه‌سازی ناهمگام (Async Mocking) داشته است.

۱. لایه‌های Domain و Persistence

تمرکز اصلی در این بخش بر روی تعریف دقیق مدل‌های داده، پیکربندی نگاشت‌ها (Fluent API)، و پیاده‌سازی عملیات پایه در Repository بوده است.

۱.۱. تعریف و مستندسازی Enumهای دامنه‌ای

`TaskPriority.cs` و **`TaskStatus.cs`** : این Enumها نقش حیاتی در تعریف چرخه‌ی حیات وظایف (Tasks) دارند. مستندسازی کامل آن‌ها برای فهم وضعیت‌های مختلف یک وظیفه انجام شده است:

- **`TaskPriority`** : شامل سطوح اهمیت (مثلاً: `Low` , `Medium` , `High` , `Critical`). در سطح پایگاه داده، این مقادیر باید با عدد صحیح (Integer) ذخیره شوند تا کارایی در مرتب‌سازی و فیلتر کردن تضمین شود.

- **TaskStatus** : نشان‌دهنده وضعیت فعلی وظیفه (مثلاً: `Pending` , `InProgress` , `Completed` , `Cancelled`). این وضعیت‌ها مسیر گردش کار (Workflow) را مشخص می‌کنند.

۱.۲. پیکربندی موجودیت `EmployeeConfiguration.cs`

پیکربندی دقیق ساختار جدول `Employees` در پایگاه داده از طریق Fluent API انجام شده است:

- محدودیت طول رشته‌ها: برای اطمینان از سازگاری با محدودیت‌های عملیاتی و بصری، طول رشته‌های مربوط به نام و نام خانوادگی به ۶۰ کاراکتر محدود شده است: `csharp builder.Property(e => e.FirstName).HasMaxLength(60).IsRequired(); builder.Property(e => e.LastName).HasMaxLength(60).IsRequired();`
- تبدیل Enum به عدد: برای ذخیره‌سازی کارآمدتر Enumها در دیتابیس، از تبدیل نوع (Type Conversion) استفاده شده است: `csharp // اگر Employee دارای وضعیت‌هایی مانند وضعیت شغلی باشد builder.Property(e => e.EmploymentStatus).HasConversion<int>().IsRequired();`
- ایندکس‌های یکتا (Unique Constraints): برای جلوگیری از تکرار داده‌های حساس در سطح پایگاه داده، ایندکس‌های یکتا بر روی فیلدهای ارتباطی اعمال شده‌اند: `csharp builder.HasIndex(e => e.Email).IsUnique(); builder.HasIndex(e => e.MobileNumber).IsUnique(false); // ممکن است MobileNumber فقط برای جستجو باشد، یا Unique باشد. در اینجا فرض بر اعمال Unique بودن است.`
- داده‌ی اولیه (Seeding): داده‌های اولیه برای اطمینان از وجود حداقل کاربران اصلی در سیستم در هنگام ساخت دیتابیس یا مهاجرت‌های اولیه تزریق شده‌اند:
  - `سروش (Soroush)` : به عنوان توسعه‌دهنده اصلی.
  - `علی کاظمی (Ali Kazemi)` : به عنوان کاربر نمونه یا مدیر اولیه.

۱.۳. بهبود `AppDbContext.cs`

- پیاده‌سازی **`IAppDbContext`** : برای تسهیل در فرآیند Mocking لایه‌ی Persistence در تست‌ها، رابط `IAppDbContext` پیاده‌سازی شده است. این امر امکان جایگزینی DbContext واقعی با یک Mock را در محیط‌های تست فراهم می‌آورد.
- بارگذاری پیکربندی‌ها: استفاده از روش استاندارد برای بارگذاری خودکار پیکربندی‌های موجود در اسمبلی‌های مرتبط: `csharp`
```
Builder.ApplyConfigurationsFromAssembly(Assembly.GetExecutingAssembly())
```
- بررسی **`()ValueGeneratedOnAdd`** : اطمینان از اینکه کلید اصلی (Primary Key) در موجودیت‌هایی که باید توسط دیتابیس تولید شوند (مانند ID وظایف)، به درستی تنظیم شده‌اند تا در هنگام ذخیره‌سازی اولیه، مقدار آن‌ها توسط EF Core تنظیم گردد: `csharp // در پیکربندی مربوطه entity.Property(e => e.Id).ValueGeneratedOnAdd();`

۱.۴. بهبود `TaskRepository.cs`

- حذف نرم (Soft Delete): متد `DeleteAsync` به جای حذف فیزیکی (Hard Delete)، عملیات حذف نرم را انجام می‌دهد. این یک الزام معماری برای حفظ تاریخچه داده‌ها است: `csharp entity.IsDeleted = true; _context.Tasks.Update(entity); await _context.SaveChangesAsync();`
- بازگرداندن شناسه تولیدشده: متد `CreateAsync` اکنون شناسه (ID) موجودیتی که تازه ایجاد شده است را پس از ذخیره در دیتابیس باز می‌گرداند. این امر برای عملیات‌های بعدی که نیاز به ID جدید دارند (مانند اضافه کردن TaskItemها به Task)، ضروری است: `csharp await _context.Tasks.AddAsync(entity); await _context.SaveChangesAsync(); // بازگرداندن ID تولید شده return entity.Id;`

---

۲. ابزارهای Mocking سازگار با EF Core 8

با توجه به تغییرات ساختاری در EF Core 8 (به ویژه در نحوه‌ی تعامل با `IQueryable` ناهمگام)، ابزارهای Mocking به روز رسانی شده‌اند.

۲.۱. `EFAsyncMockHelper.cs` (با نام جدید `CreateMockDbSet`)

این ابزار برای شبیه‌سازی `DbSet<TEntity>` به شکلی که متدهای ناهمگام EF Core را پشتیبانی کند، طراحی شده است:

- متد **`AddAsync`** : شبیه‌سازی شده تا داده‌ها را به لیست محلی اضافه کند.
- متد **`FindAsync`** : پیاده‌سازی شده برای جستجوی سریع در لیست شبیه‌سازی شده.

- پشتیبانی از **`IAsyncEnumerable`** : مهم‌ترین بخش، پیاده‌سازی صحیح برای پشتیبانی از متدهای ناهمگام LINQ (مانند `ToListAsync()`) است. این کار با استفاده از پیاده‌سازی‌های سفارشی برای `IAsyncEnumerator` و `IAsyncQueryProvider` انجام می‌شود.

- پشتیبانی از نسخه‌های قدیمی‌تر: متد کمکی `BuildMockDbSet` همچنان حفظ شده تا با پروژه‌هایی که از نسخه‌های قدیمی‌تر EF Core استفاده می‌کنند، سازگاری داشته باشد.

۲.۲. پیاده‌سازی شبیه‌سازی ناهمگام

کلاس‌های کمکی زیر برای شبیه‌سازی رفتار واقعی EF Core در محیط تست ایجاد و تثبیت شده‌اند:

- **`TestAsyncEnumerable.cs`** , **`TestAsyncEnumerator.cs`** : این کلاس‌ها کلاس‌های پایه برای اجرای کوئری‌ها به صورت ناهمگام هستند. آن‌ها اطمینان می‌دهند که اجرای متدهایی مانند `await query.ToListAsync()` در تست‌ها به درستی کار کند.
- **`TestAsyncQueryProvider.cs`** : مسئولیت اجرای دستورات LINQ (مانند `Where` , `Select`) بر روی مجموعه‌ی داده‌ی شبیه‌سازی شده را بر عهده دارد و نتیجه را به شکل یک `IQueryable` شبیه‌سازی شده بازمی‌گرداند.
- توضیحات **`ExecuteAsync<TResult>`** : مستندسازی شده است که نحوه‌ی فراخوانی و تفسیر نتایج توسط EF Core 8 در این بخش چگونه است و چگونه Mock ما باید این رفتار را تقلید کند.

---

۳. تست‌های واحد (Unit Tests)

تست‌های واحد بر روی منطق کسب‌وکار (Business Logic) و مدیریت استثناها تمرکز دارند.

۳.۱ `BusinessExceptionTests.cs`

تست‌های سازنده‌های استثنای سفارشی کسب‌وکار (مانند `DomainValidationException`):

- تست سازنده‌های مختلف: اطمینان از اینکه استثناها با یک پیام پیش‌فرض یا یک پیام سفارشی تولید می‌شوند.
- اطمینان از فرمت پیام فارسی: تست شده است که پیام‌های استثنا به درستی با استفاده از کاراکترهای یونیکد فارسی نمایش داده می‌شوند و قالب‌بندی آن‌ها استاندارد است.

۳.۲ `NotFoundExceptionTests.cs`

این تست‌ها بر روی استثنای `NotFoundException` که هنگام عدم یافتن یک موجودیت در سرویس‌ها یا ریپوزیتوری‌ها پرتاب می‌شود، تمرکز دارد.

- تست قالب استاندارد پیام: اطمینان از تولید قالب پیام دقیق: > «موجودیت 'X' با شناسهٔ 'Y' یافت نشد.»
- پشتیبانی از انواع کلید (Key Types): تست شده است که این الگو برای کلیدهای از نوع `int` و همچنین کلیدهای از نوع `string` (مانند GUID یا کد محصول) به درستی اعمال شود.

---

۴. تست ادغام (Integration Test)

تست‌های ادغام بر روی تعامل صحیح بین لایه‌ها (Mapper، Service، Controller، Repository) در یک محیط شبیه‌سازی شده‌ی کامل تمرکز دارند.

۴.۱. `TaskServiceIntegrationTests.cs`

این تست‌ها شامل سناریوهای کامل CRUD برای مدیریت وظایف هستند:

- ساخت DI Container: استفاده از `Microsoft.Extensions.DependencyInjection` برای ساخت یک کانتینر تزریق وابستگی (DI Container) سبک وزن در محیط تست.
  - DbContext InMemory: DbContext با Provider حافظه (In-Memory Database) پیکربندی شده تا سرعت تست بالا رود و داده‌ها پس از هر تست پاک شوند.
  - AutoMapper: تزریق پروفایل‌های نگاشت (Mapping Profiles) برای اطمینان از تبدیل صحیح DTOها به Domain و بالعکس.
  - Validators: تزریق کتابخانه‌ی اعتبارسنجی (مانند FluentValidation) برای تست صحت ورودی‌ها.
  - MediatR: تزریق Handlerها برای اطمینان از اجرای صحیح زنجیره‌ی Command/Query.
- اجرای تست‌های CRUD کامل: تست Create، Read (تکی و لیستی)، Update و Delete (نرم) بر روی سرویس `TaskService` .
- متد **`Cleanup`** : پیاده‌سازی متد آزادسازی منابع تست، اگرچه در DbContext InMemory معمولاً نیاز به آزادسازی صریح نیست، اما برای رعایت اصول، تزریق منابع غیرضروری بررسی شده است.

---

۵. فایل عمومی و تنظیمات تست‌ها

۵.۱. `GlobalUsings.cs`

برای کاهش تکرار در فایل‌های تست (که معمولاً حجیم هستند)، یک فایل متمرکز برای `using` های مشترک ایجاد شده است:

- متمرکزسازی فضای نام‌های تست: شامل `Xunit` (برای چارچوب تست)، `Microsoft.EntityFrameworkCore` (برای Mocking EF Core)، و `FluentAssertions` (برای Assertions شی‌گرا).
- ساده‌سازی پیکربندی: این امر باعث می‌شود هر فایل تست تنها بر روی منطق خود متمرکز شود و نیازی به تکرار دستورات `using` نباشد.

جمع‌بندی نهایی

تمام اجزای فنی پروژه، از مدل‌سازی دیتابیس گرفته تا شبیه‌سازی ناهمگام برای تست، مورد بازبینی و تثبیت نهایی قرار گرفتند. این نسخه شامل بهبودهای مهمی در پایداری تست‌ها و رعایت استانداردهای مدرن EF Core 8 است.

تمام اجزا با سبک آموزشی RiRaDocs Teaching Edition مستندسازی شدند. این نسخه‌ی نهایی برای آموزش، توسعه، و مرجع رسمی پروژه RiRa محسوب می‌شود و هم‌اکنون آماده‌ی انتشار با تگ:

`RiraDocs-v2025.11.4-Stable-Final-Fixed`

تهیه‌کننده: سروش (RiRaDocs Maintainer) تاریخ تولید: ۴ نوامبر ۲۰۲۵ 🏷️ فرمت: PDF (نسخه‌ی رسمی جهت بایگانی داخلی RiRa)