**مثالی از CQRS وEvent Sourcing :**

**این پروژه یک وب سرویس است برای انجام کارهایی مربوط به حساب مانند: ایجاد حساب ،برداشت و واریز و گرفتن مانده نوشت شده است ضنما این پروژه با تکنولووژی asp.net .net core 8 با دیتابیس SQLLite توسعه یافت است .**

**بخش اول - توضیحات جامع درباره CQRS (Command Query Responsibility Segregation) و Event Sourcing :**

**CQRS (تفکیک مسئولیت‌های فرمان و پرسش)**

**CQRS یک الگوی معماری است که در آن وظایف مربوط به خواندن داده‌ها و نوشتن داده‌ها از یکدیگر جدا می‌شوند. این الگو به طور ویژه در سیستم‌هایی که مقیاس‌پذیری و عملکرد بالا نیاز دارند، سودمند است.**

**اصول CQRS:**

1. **تفکیک وظایف:**
   * **Commands: وظیفه آن‌ها تغییر وضعیت سیستم است. این‌ها اغلب شامل عملیات‌هایی مانند ایجاد، به‌روزرسانی یا حذف داده‌ها هستند.**
   * **Queries: وظیفه آن‌ها بازیابی داده‌ها از سیستم است. این‌ها عملیات‌هایی هستند که به داده‌ها دسترسی دارند بدون تغییر وضعیت آن‌ها.**
2. **مدل‌های مختلف داده:**
   * **در CQRS، معمولاً از مدل‌های متفاوتی برای خواندن و نوشتن داده‌ها استفاده می‌شود. این می‌تواند به بهینه‌سازی عملکرد و مقیاس‌پذیری کمک کند.**
   * **برای مثال، مدل نوشتن ممکن است کاملاً نرمال‌سازی شده باشد، در حالی که مدل خواندن می‌تواند انباشته شده و بهینه باشد.**
3. **بهبود مقیاس‌پذیری:**
   * **با جدا کردن وظایف خواندن و نوشتن، می‌توان هر کدام را به صورت جداگانه مقیاس‌پذیر کرد، به عنوان مثال، با استفاده از بارگذاری متوازن یا کش.**

**مزایای CQRS:**

* **مقیاس‌پذیری بالا: امکان مقیاس‌گذاری مستقل بخش‌های فرمان و پرسش.**
* **تنظیم بهینه عملکرد: بهینه‌سازی جداگانه برای خواندن و نوشتن.**
* **ساده‌سازی کد: درک بهتر ساختار سیستم و تسهیل نگهداری کد.**
* **قابلیت تغییر: تحمل پذیرتر در برابر تغییرات کسب‌وکار و به روز رسانی‌های آینده.**

**معایب CQRS:**

* **پیچیدگی بیشتر: جداسازی بخش‌های خواندن و نوشتن می‌تواند پیچیدگی معماری را افزایش دهد.**
* **نیاز به هماهنگی: نیاز به هماهنگی بین دو بخش در صورت تغییر وضعیت سیستم.**

**Event Sourcing (ذخیره‌سازی رویداد)**

**Event Sourcing یک الگوی معماری است که در آن وضعیت سیستم به واسطه رویدادهایی که باعث تغییر آن وضعیت شده‌اند، ذخیره می‌شود. به عبارت دیگر، به جای ذخیره وضعیت کنونی سیستم، تمام تغییرات (رویدادها) ثبت می‌شوند.**

**اصول Event Sourcing:**

1. **ذخیره‌سازی رویدادها:**
   * **تمامی تغییرات به عنوان مجموعه‌ای از رویدادها ذخیره می‌شوند. هر رویداد نمایانگر یک تغییر در وضعیت سیستم است.**
   * **این رویدادها به ترتیب زمان ثبت می‌شوند و می‌توانند به راحتی بازسازی وضعیت کنونی سیستم را فراهم کنند.**
2. **بازسازی وضعیت:**
   * **برای به‌دست آوردن وضعیت جاری سیستم، باید تمام رویدادها را از ابتدا تا کنون بروزرسانی کرد. این کار با replay کردن رویدادها امکان‌پذیر است.**
3. **تضمین یکپارچگی:**
   * **از آنجا که هر تغییر در سیستم به صورت یک رویداد ثبت می‌شود، این باعث می‌شود که تاریخچه کاملی از تغییرات در دسترس باشد، که این خود موجب افزایش شفافیت و یکپارچگی می‌شود.**

**مزایای Event Sourcing:**

* **تاریخچه کامل تغییرات: امکان بررسی تمامی تغییرات و وضعیت‌های پیشین.**
* **آسانی در بازیابی و تنقیح داده‌ها: بازسازی وضعیت‌ها در صورت نیاز.**
* **امکان پردازش‌های پیچیده: جداسازی لایه‌ها و استفاده از پردازش‌های متفاوت بر روی رویدادها.**

**معایب Event Sourcing:**

* **پیچیدگی بیشتر: نیاز به مدیریت و پردازش تعداد زیادی رویداد.**
* **اندازه بزرگ داده‌ها: ذخیره تمام رویدادها می‌تواند منجر به حجم بالای داده شود.**
* **نقص در تفسیر رویدادها: نیاز به استاندارد سازی و تفسیر صحیح رویدادها برای اطمینان از درک درست آن‌ها.**

**رابطه بین CQRS و Event Sourcing**

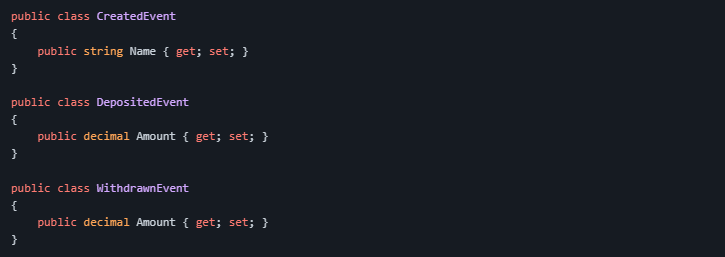
**CQRS و Event Sourcing معمولاً با هم ترکیب می‌شوند. در این ترکیب، می‌توان رویدادها را به عنوان روش ذخیره‌سازی داده‌ها در بخش نوشتن CQRS استفاده کرد. وقتی که دستوراتی به سیستم وارد می‌شوند، رویدادهای مربوط به آن دستور ذخیره می‌شوند و از طریق آن‌ها می‌توان وضعیت کنونی را بازسازی کرد.**

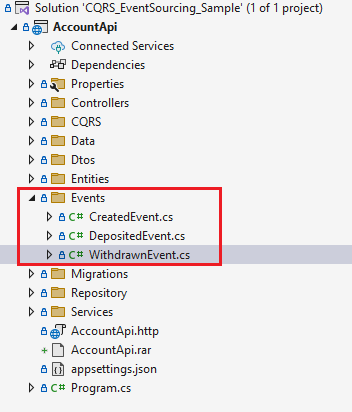
**بخش دوم -استفاده از CQRS و Event Sourcing، ما به یک برنامه مدیریت حساب‌های بانکی خواهیم پرداخت. در این مثال، ساختار برنامه شامل بخش‌های زیر است:**

1. **مدل رویداد (Events): شامل رویدادها مانند واریز، برداشت، و ایجاد حساب.**
2. **ایجاد Commands و Queries :**
   1. **دستورات (Commands): شامل عملیاتی برای تغییر وضعیت سیستم، مانند واریز و برداشت.**
   2. **پرسش‌ها (Queries): شامل درخواست‌های برای دریافت داده‌ها، مانند دریافت موجودی حساب.**
3. **حساب (Account): پیاده‌سازی حساب که وضعیت را بر اساس رویدادها پیگیری می‌کند.**
4. **مدل event**
5. **ذخیره (Repository) حساب ها**
6. **پیاده سازی پردازش Commands و Queries – برای ذخیره حساب و رویدادها**
7. **مدیریت رویداد**
8. **api**

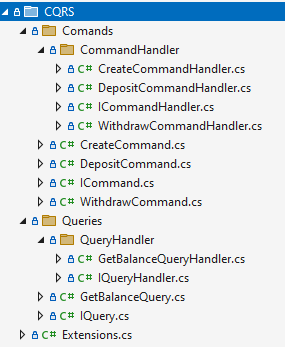
**1. تعریف مدل‌های رویداد**

**مدل رویداد برای ذخیره‌سازی تغییرات وضعیت:**

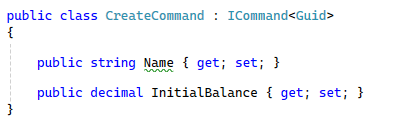


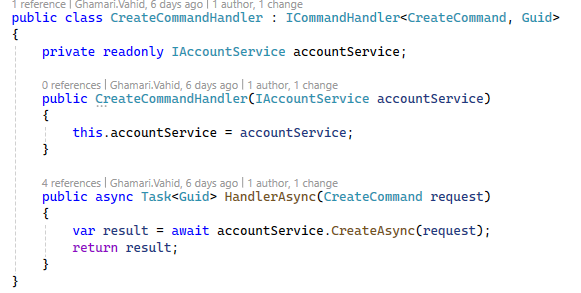
****

**2. ایجاد دستورات و پرسش‌ها**

****

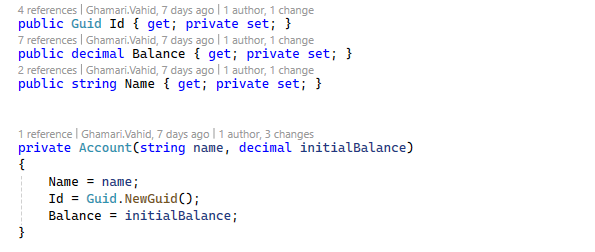
**به عنوان مثال برای ایجاد حساب**

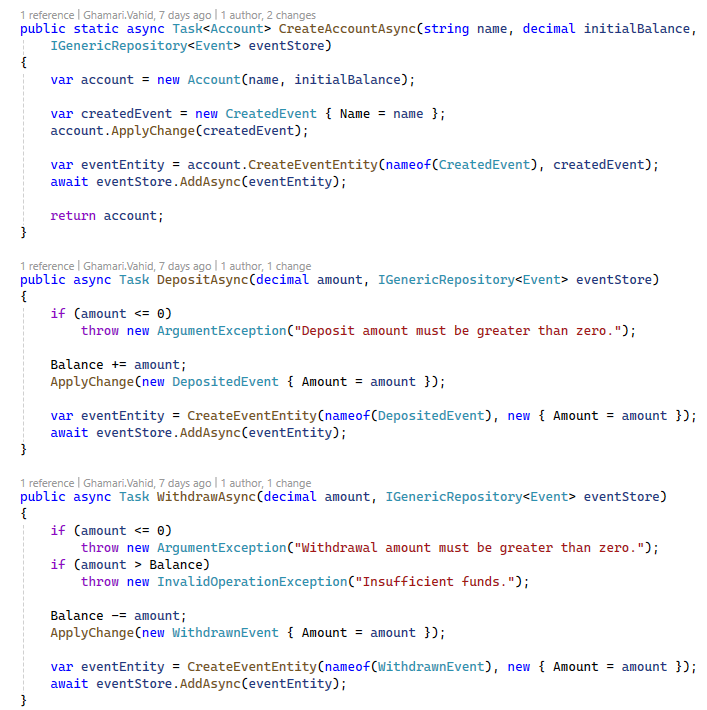
****

****

**3. پیاده‌سازی حساب**

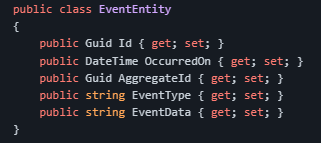
**کلاس Account که شامل منطق CQRS و Event Sourcing است:**

****

****

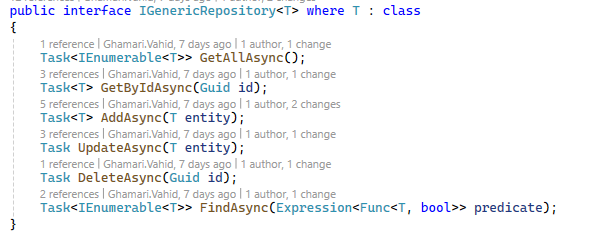
****

**4- رویدار:در اینجا مقدار هر event و تایپ اون رو ذخیره میکنم**

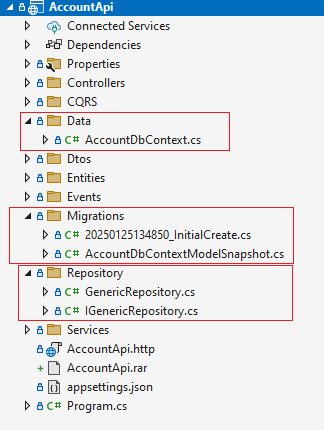
****

**5-ریپازتوری**

**این ریپازتوری به شکل generic پیاده سازی شده تا هم event وهم ذخیره حساب انجام شود.**

****

****

****

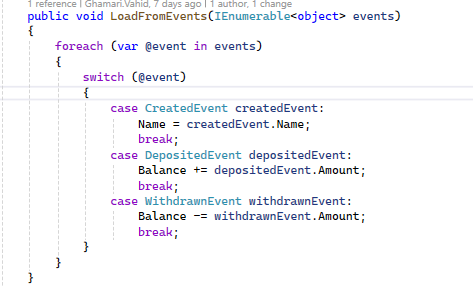
**6- پیاده‌سازی پردازش دستورات و پرسش‌ها در یک سرویس**

****

**7-مدیریت رویدادها**

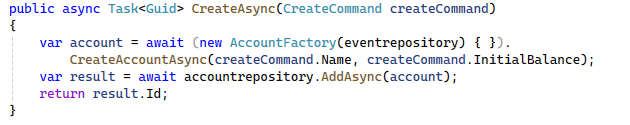
**متد LoadFromEvents به‌منظور بارگذاری و بازیابی وضعیت داخلی کلاس Account از روی رویدادهایی که در طول زمان ذخیره شده‌اند، طراحی شده است. معمولاً این متد باید در درون کلاس مدل اصلی که نمایانگر موجودیت شماست، به نام Account در این مورد، تعریف شود.**

**شما باید متد LoadFromEvents را به کلاس Account اضافه کنید. این متد به هر رویدادی که به حساب مربوط می‌شود پاسخ می‌دهد و وضعیت Account را بر اساس آن رویدادها به‌روزرسانی می‌کند.**

****

**استفاده از متد LoadFromEvents**

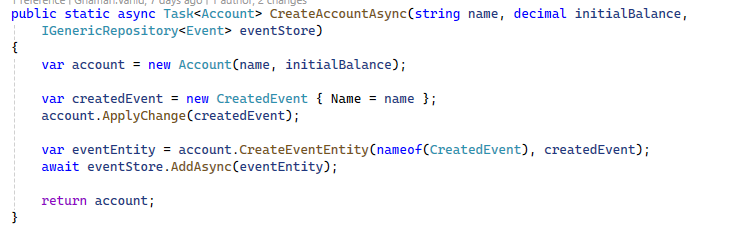
**وقتی شما یک حساب را بارگذاری می‌کنید، دیگر نیاز نیست که به تمام رویدادها روی بیاورید. می‌توانید از متد LoadFromEvents برای به‌روزرسانی وضعیت حساب به‌کار ببرید. به این شکل:**

****

**این ترتیب به شما این امکان را می‌دهد که وضعیت حساب را در هنگام بارگذاری از روی رویدادها به‌راحتی مدیریت کنید. هر بار که یک حساب بارگذاری می‌شود، وضعیت آن بر اساس رویدادهای ذخیره‌شده به‌روز می‌شود.**

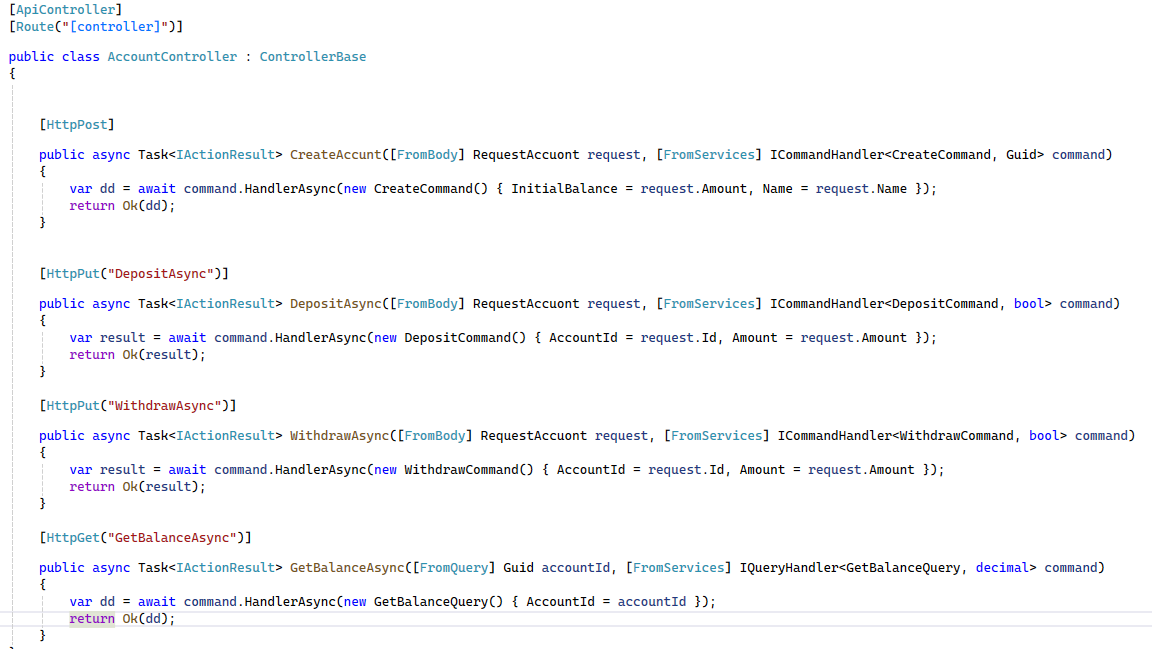
**نکته : ما در این پروزه از AccountFactory استفاده کرده ایم زیرا لازم بود در زمان ایجاد از سرویس های استفاده کنم که async بوده :**

****

**دیگر ما contractor به شکل public نداریم **

* **اگر فقط در حال ایجاد حساب هستید و رویدادهای جدید را ذخیره می‌کنید، نیازی به LoadFromEvents نیست.**
* **اگر بخواهید یک حساب موجود را از روی رویدادهایش بازسازی کنید، می‌توانید از LoadFromEvents استفاده کنید و این متد مرتبط با بارگذاری وضعیت حساب از روی رویدادها خواهد بود.**

**8- api**

****

**با این طراحی:**

* **CQRS: شما دستورات را از پرسش‌ها جدا کرده‌اید.**
* **Event Sourcing: با استفاده از رویدادها وضعیت سیستم را بازیابی می‌کنید.**
* **می‌توانید به‌راحتی تغییرات را پیگیری کرده و منطق پیچیده‌تری را ایجاد کنید.**

**نکته تکمیلی Program**

****

**بخش سوم :**

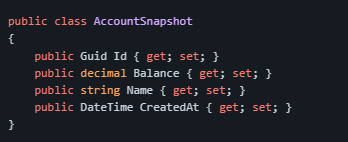
**در پیاده‌سازی‌ای که انجام شد، فقط رویدادها ذخیره می‌شوند و وضعیت کامل حساب (Account) به‌طور مستقیم ذخیره نمی‌شود. به همین دلیل، هر بار که می‌خواهید در مورد حساب‌ها اطلاعات جدیدی کسب کنید، باید همه رویدادهای مربوط به آن حساب را بارگذاری کنید و از آن‌ها برای بازسازی وضعیت استفاده کنید.**

**برای کامل کردن این فرآیند، می‌توانیم به دو روش پیش برویم:**

**1- ذخیره وضعیت حساب (Snapshot)**

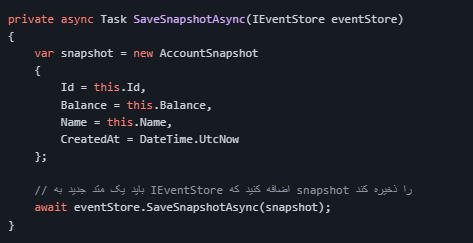
**در این روش، علاوه بر ذخیره‌سازی رویدادها، یک snapshot از وضعیت فعلی حساب نیز ذخیره می‌کنیم تا بازیابی سریع‌تری انجام دهیم. این کار می‌تواند عملکرد را بهبود بخشد، مخصوصاً اگر تعداد زیادی رویداد مربوط به یک حساب وجود داشته باشد.**

**1- تعریف مدل Snapshot**



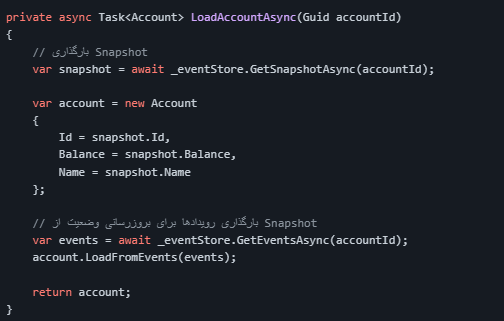
**2- ذخیره Snapshot پس از تشکیل رویداد**

**در متد DepositAsync و WithdrawAsync، پس از ذخیره‌سازی رویداد، می‌توانید Snapshot را نیز ذخیره کنید. اگر از تنظیمات سخت‌افزاری یا نرم‌افزاری خاصی استفاده می‌کنید، پیشنهاد می‌شود در یک دوره معین (مثلاً هر ۵۰ رویداد)، Snapshot را ذخیره کنید.**



**3-کار با Snapshot**

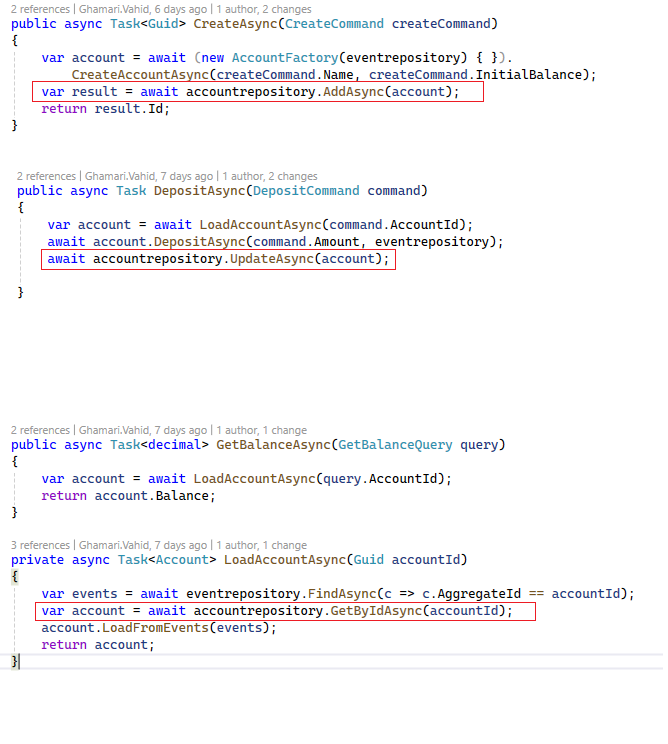
**وقتی که وضعیت حساب از روی رویدادها بارگذاری می‌شود، ابتدا می‌توانید از Snapshot استفاده کنید:**



**2- ایجاد یک منبع داده برای حساب‌ها**

**می‌توانیم یک منبع داده (مانند پایگاه داده) برای ذخیره‌سازی وضعیت فعلی حساب‌ها داشته باشیم. در این حالت، آنچه ذخیره می‌کنیم، رویدادها هستند و در صورت نیاز، وضعیت حساب‌ها از روی رویدادها بازیابی می‌شود.**

**در پروزه که انجام دادیم ما از این روش استفاده کردم مثال :**

****