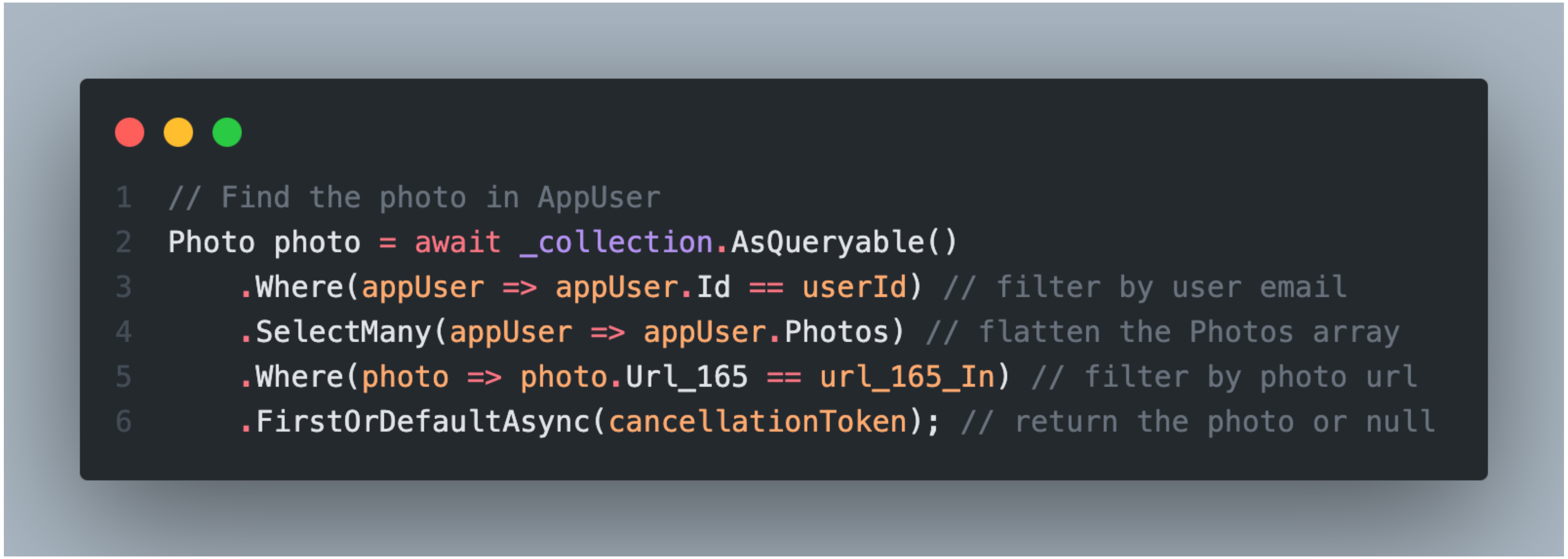
“Shoot for the stars, so if you fall you land on a cloud.”

Kanye West

AsQueryable

What is that?

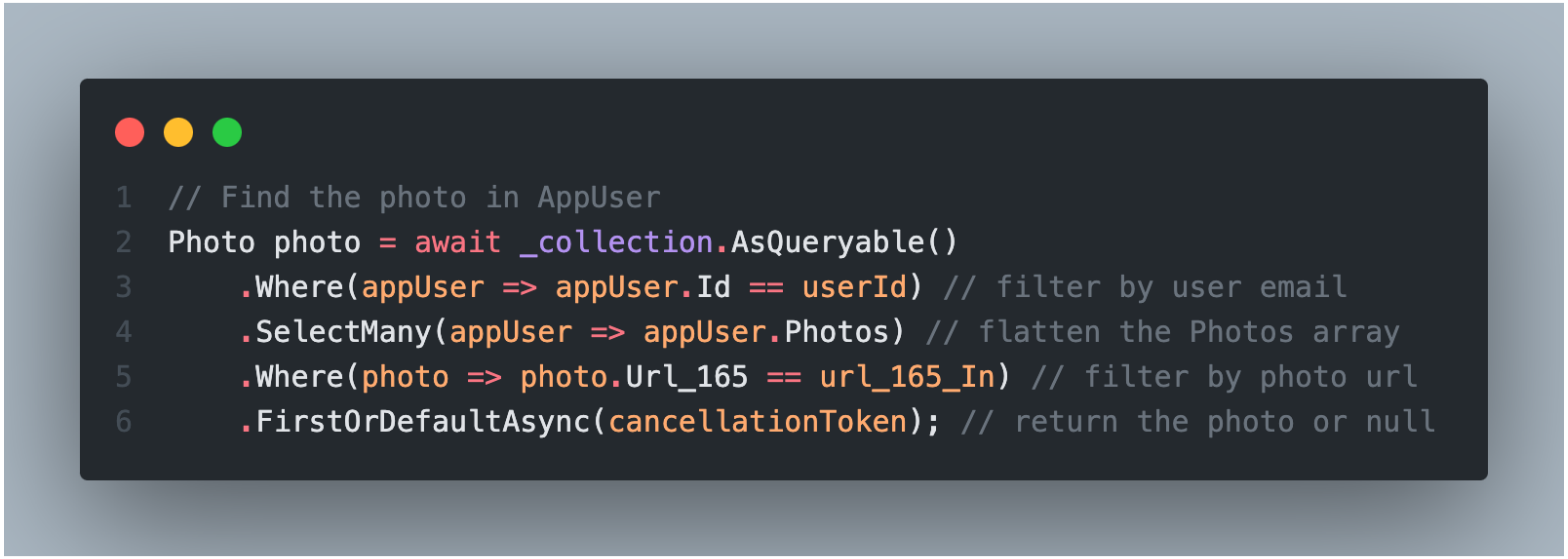
Enables LINQ Querying: Once the collection is represented as IQueryable<T>, it enables the use of LINQ operators and extensions methods that are specific to IQueryable<T>. These include methods like Where(), Select(), OrderBy(), and others that can be used to query data in a more flexible and efficient way compared to standard LINQ to Objects operations.



```
// Find the photo in AppUser
Photo photo = await _collection.AsQueryable()
    .Where(appUser => appUser.Id == userId) // filter by user email
    .SelectMany(appUser => appUser.Photos) // flatten the Photos array
    .Where(photo => photo.Url_165 == url_165_In) // filter by photo url
    .FirstOrDefaultAsync(cancellationToken); // return the photo or null
```

.Where?

.Where(appUser => appUser.Id == userId): This part filters the collection to include only elements where the Id property of the appUser matches the userId provided. This is essentially a condition applied to filter out specific elements from the collection based on the user's ID.



```
// Find the photo in AppUser
Photo photo = await _collection.AsQueryable()
    .Where(appUser => appUser.Id == userId) // filter by user email
    .SelectMany(appUser => appUser.Photos) // flatten the Photos array
    .Where(photo => photo.Url_165 == url_165_In) // filter by photo url
    .FirstOrDefaultAsync(cancellationToken); // return the photo or null
```

.SelectMany?

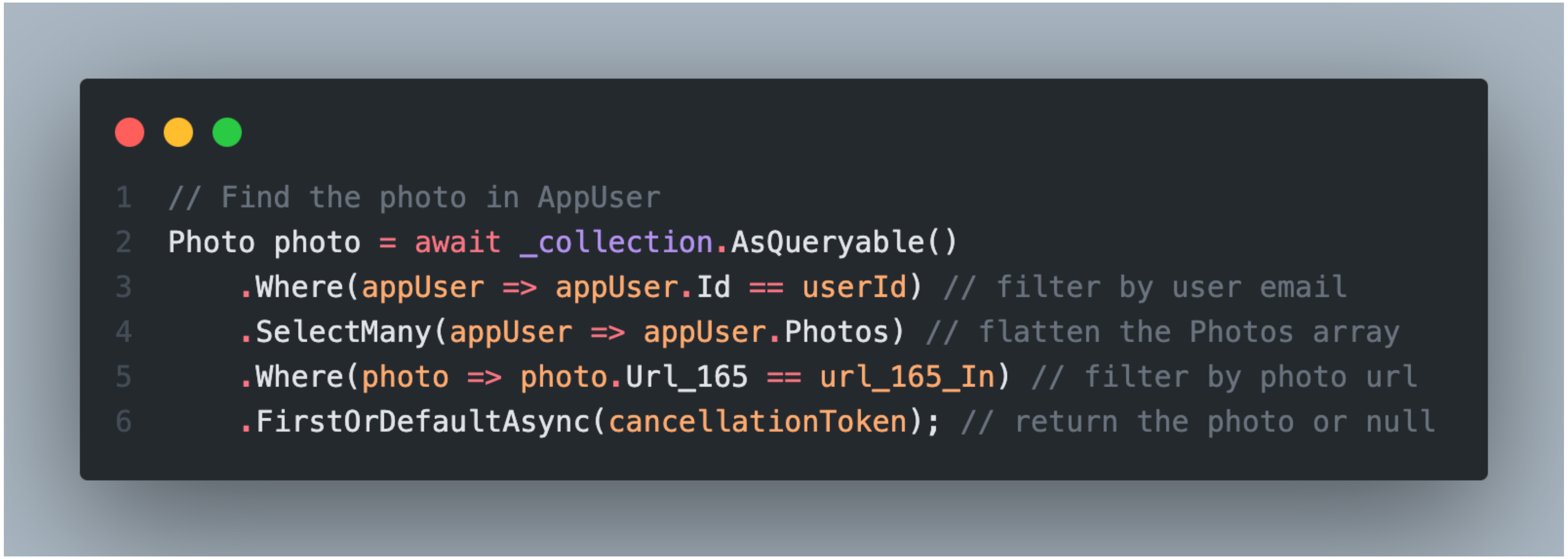
It flattens the resulting sequence of photo collections into one sequence. Essentially, it's unwrapping nested collections of photos into a single collection of photos.

Imagine you have a collection of users, and each user has a collection of photos. The original data might look something like this:

```
User1 -> [Photo1, Photo2, Photo3]
User2 -> [Photo4, Photo5]
User3 -> [Photo6]
```

After SelectMany:

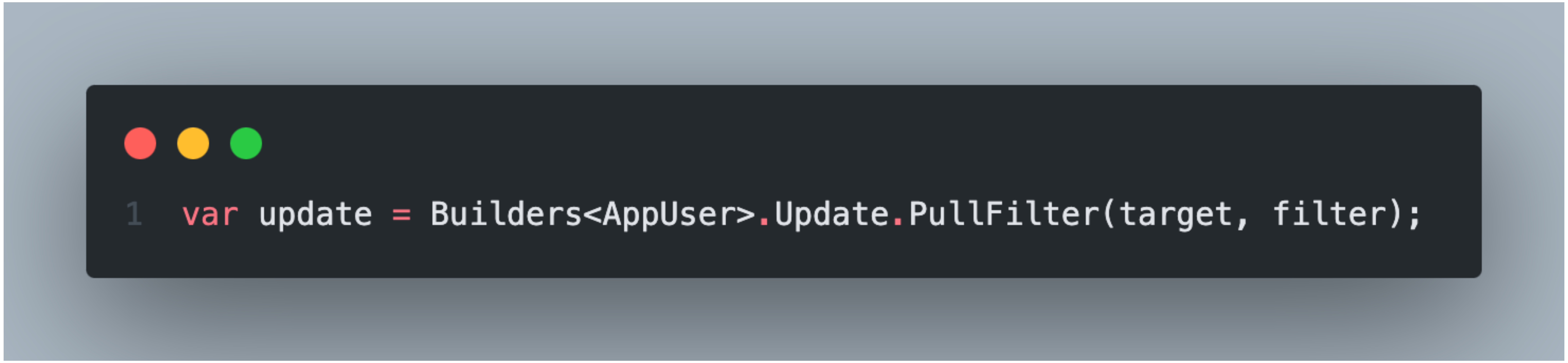
```
[Photo1, Photo2, Photo3, Photo4, Photo5, Photo6]
```



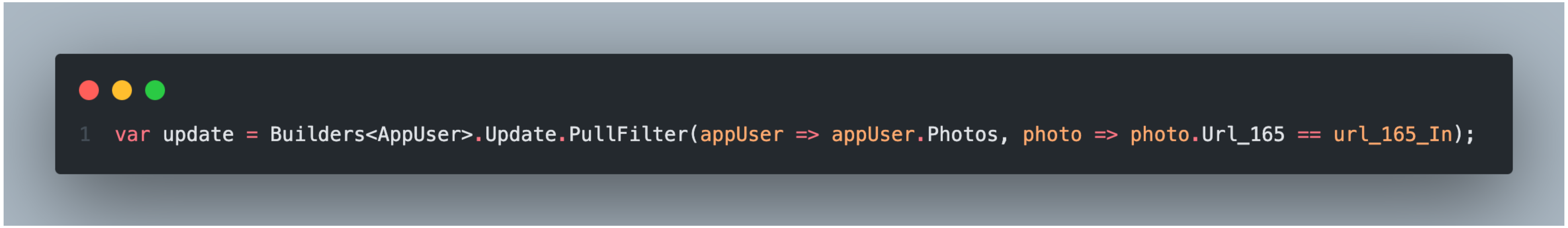
```
// Find the photo in AppUser
Photo photo = await _collection.AsQueryable()
    .Where(appUser => appUser.Id == userId) // filter by user email
    .SelectMany(appUser => appUser.Photos) // flatten the Photos array
    .Where(photo => photo.Url_165 == url_165_In) // filter by photo url
    .FirstOrDefaultAsync(cancellationToken); // return the photo or null
```

Pull Filter

How it works?



```
var update = Builders<AppUser>.Update.PullFilter(target, filter);
```



```
var update = Builders<AppUser>.Update.PullFilter(appUser => appUser.Photos, photo => photo.Url_165 == url_165_In);
```