

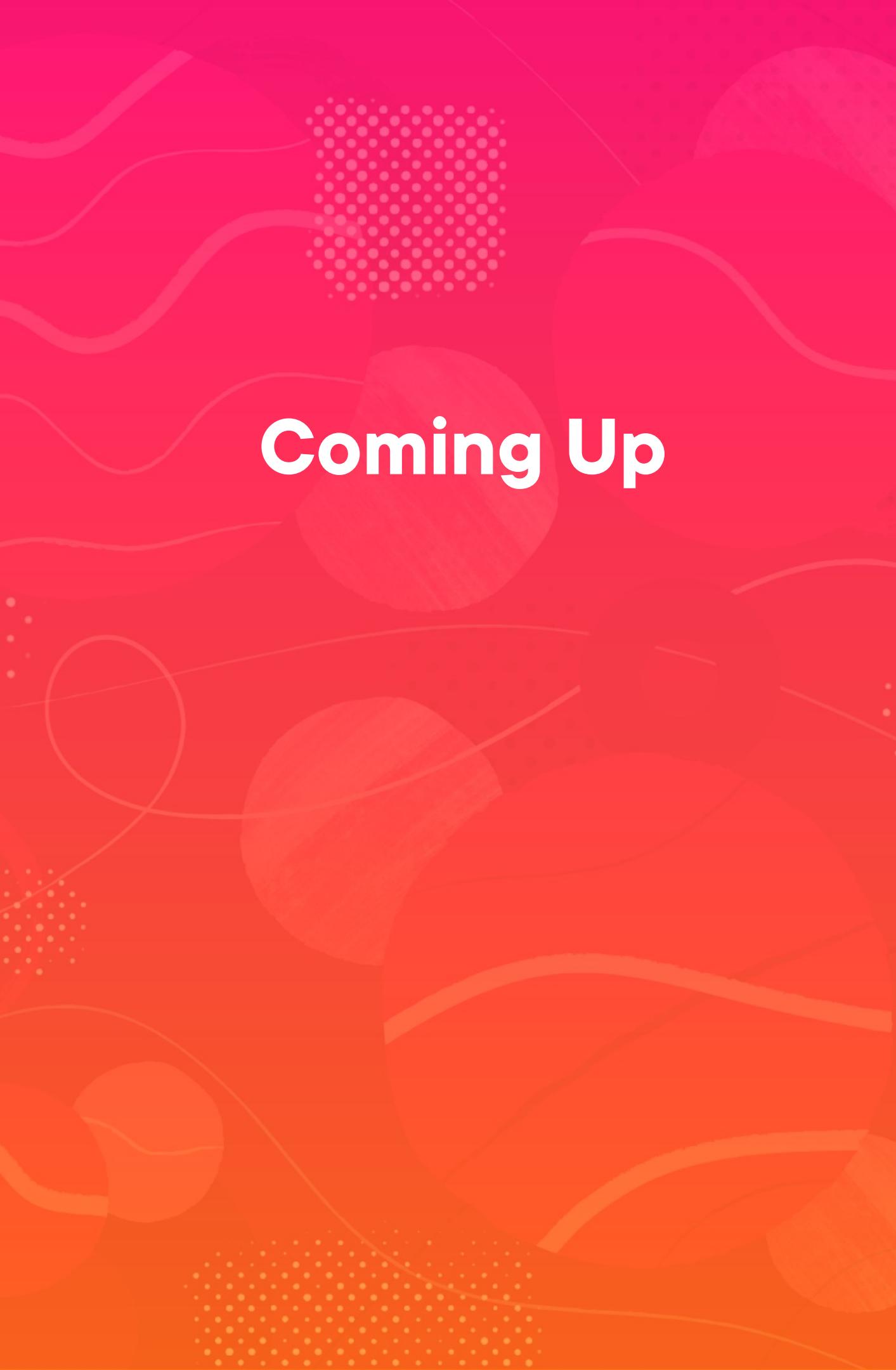
Dealing with Asynchronous Service Integrations and Supporting Cancellation



Kevin Dockx

Architect

@Kevindockx | www.kevindockx.com



Coming Up

Integrating with an external service

Dealing with multiple service calls

- Asynchronously
- Parallel

Supporting task cancellation

Gracefully handling exceptions



Demo

Asynchronously integrating with an external service



Demo

**Processing multiple service calls
asynchronously, one by one**



Demo

**Processing multiple service calls
asynchronously, after waiting for all of
them to complete**



Parallel Processing Vs. Asynchronous Processing

Using `async await` all the way through

- Threads potentially get freed up
- Scalability is improved



Parallelism

At least two threads are executing simultaneously



```
Parallel.ForEach(bookCoverUrls,  
    bookCoverUrl => httpClient.GetAsync(bookCoverUrl));
```

Parallel Processing

Threads are blocked

Negative impact on performance due to thread exhaustion

Code like this should not be used on a server



```
var bookCoverTasks = new List<Task<HttpResponseMessage>>();  
foreach (var bookCoverUrl in bookCoverUrls)  
{  
    bookCoverTasks.Add(httpClient.GetAsync(bookCoverUrl));  
};  
  
Task.WaitAll(bookCoverTasks.ToArray());
```

Wait with Task.WaitAll

The current thread is being blocked

Negative impact on performance due to thread exhaustion

Code like this should not be used on a server



Demo

**Passing multiple objects to a result filter
with ValueTuple**



Demo

Mapping multiple objects into one



Why Supporting Cancellation Matters

Frees up threads (I/O-bound work)

- Improves scalability

Frees up CPU resources (computational-bound work)



CancellationTokenSource

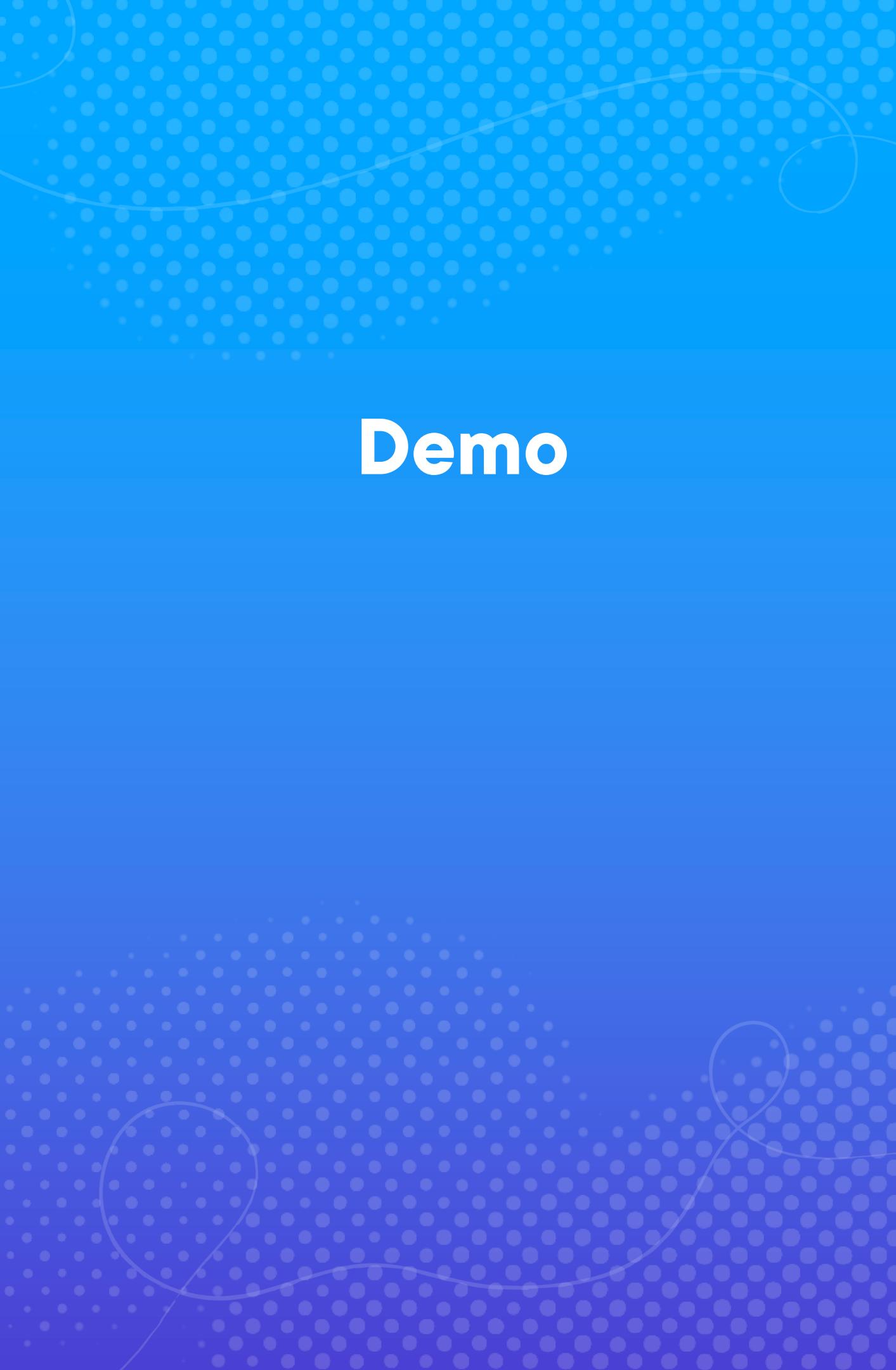
An object which manages cancellation tokens and sends cancellation notifications to the individual cancellation tokens



Cancellation token

A lightweight value type passed to one or more listeners, typically as a method parameter





Demo

Supporting cancellation



Demo

Supporting cancellation when the consumer navigates away



Demo

Listening to multiple cancellation tokens



Handling Exceptions in Async Code



A try/catch-block surrounding
`Task.Run` would not catch
exceptions that happened inside of
that statement

When working with multiple tasks,
exceptions would be wrapped in an
`AggregateException`



Handling Exceptions in Async Code

Using `async await` all the way through solves
these two issues



Handling Exceptions in Async Code



Catch the exception close to where it occurred with a try/catch-block



Write an `ExceptionFilter`



Configure the `ExceptionHandler` middleware



...



Handling Exceptions in Async Code

Important to remember is that you should catch an `OperationCancelledException`

- `TaskCancelledException` derives from it



Summary

Execute multiple tasks asynchronously, not in parallel

- `async await`
- `await WhenAll(...), await WhenAny(...)`



Summary

CancellationTokenSource and cancellation tokens

- Tokens that are action parameters will receive a notification when the consumer navigates away



Summary

**Catch an OperationCancelledException,
TaskCancelledException derives from it**



Additional Return Types and Avoiding Common Pitfalls

