



Wow, this is a lot of boilerplate to publish/receive a message!

A black arrow pointing downwards from the first box to the second box.

Let's try to refactor this to make it much easier to publish/receive

A black arrow pointing downwards from the second box to the third box.

We'll write out an initial implementation in this test project, then move it to our common module

Class Listener			
	Property	Type	Goal
abstract	subject	string	Name of the channel this listener is going to listen to
abstract	onMessage	(event: EventData) => void	Function to run when a message is received
	client	Stan	Pre-initialized NATS client
abstract	queueGroupName	string	Name of the queue group this listener will join
	ackWait	number	Number of seconds this listener has to ack a message
	subscriptionOptions	SubscriptionOptions	Default subscription options
	listen	() => void	Code to set up the subscription
	parseMessage	(msg: Message) => any	Helper function to parse a message

Class Listener

```
graph TD; A[Class Listener] --> B[Class TicketCreatedListener]; A --> C[Class OrderUpdatedListener];
```

**Class
TicketCreatedListener**

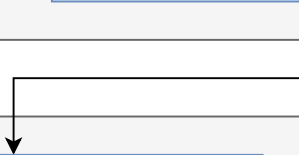
*Listens specifically for
'ticket:created' events*

**Class
OrderUpdatedListener**

*Listens specifically for
'order:updated' events*

Defined in common module

Class Listener



**Class
TicketCreatedListener**

Defined in payments service

**Class
OrderUpdatedListener**

Defined in tickets service

**There is a strong mapping between
subject names and event data**

ticket:created



Event Data

<i>key</i>	<i>type</i>
id	string
title	string
price	number

order:updated



Event Data

<i>key</i>	<i>type</i>
id	string
userId	string
ticketId	string

Class TicketCreatedListener

<i>Property</i>	<i>Type</i>
subject	string
onMessage	(data: any) => void

ticket:created

Event Data for 'ticket:created'

<i>key</i>	<i>type</i>
id	string
title	string
price	number

Mismatch! Should result in an error somewhere!

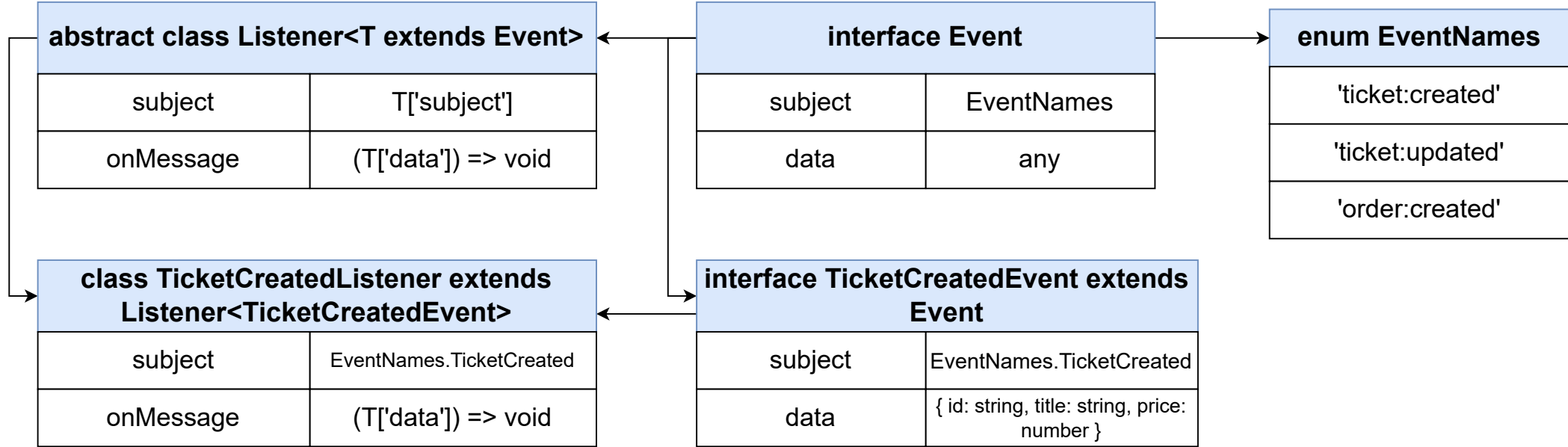
Class TicketCreatedListener

<i>Property</i>	<i>Type</i>
subject	string
onMessage	(data: any) => void

order:updated

Event Data for 'ticket:created'

<i>key</i>	<i>type</i>
id	string
title	string
price	number



event-subjects.ts

enum EventSubjects

'ticket:created'

'ticket:updated'

'order:created'

ticket-created-listener.ts

interface TicketCreatedEvent

subject

EventSubjects.TicketCreated

data

{ id: string, title: string, price:
number }

class TicketCreatedListener extends Listener<TicketCreatedEvent>

subject

EventSubjects.TicketCreated

onEvent

(data:
TicketCreatedEvent['data']) =>
void

Defined in common module

enum Subjects

Class Listener

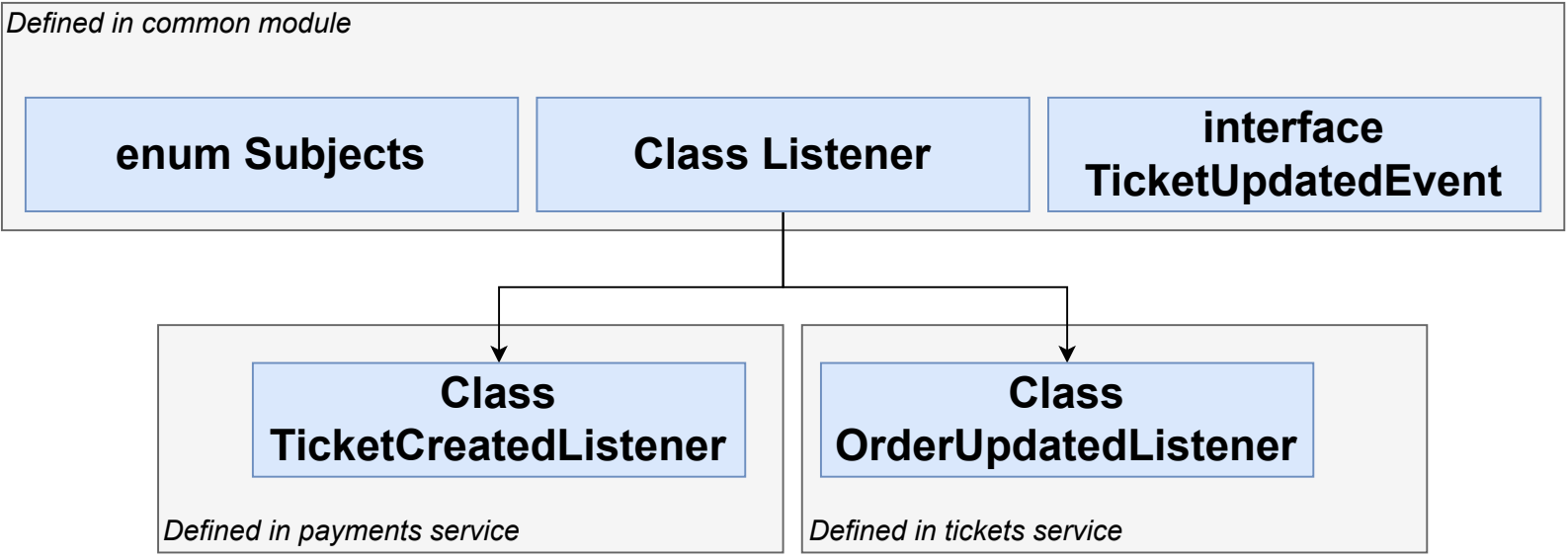
**interface
TicketUpdatedEvent**

**Class
TicketCreatedListener**

Defined in payments service

**Class
OrderUpdatedListener**

Defined in tickets service



Publishers

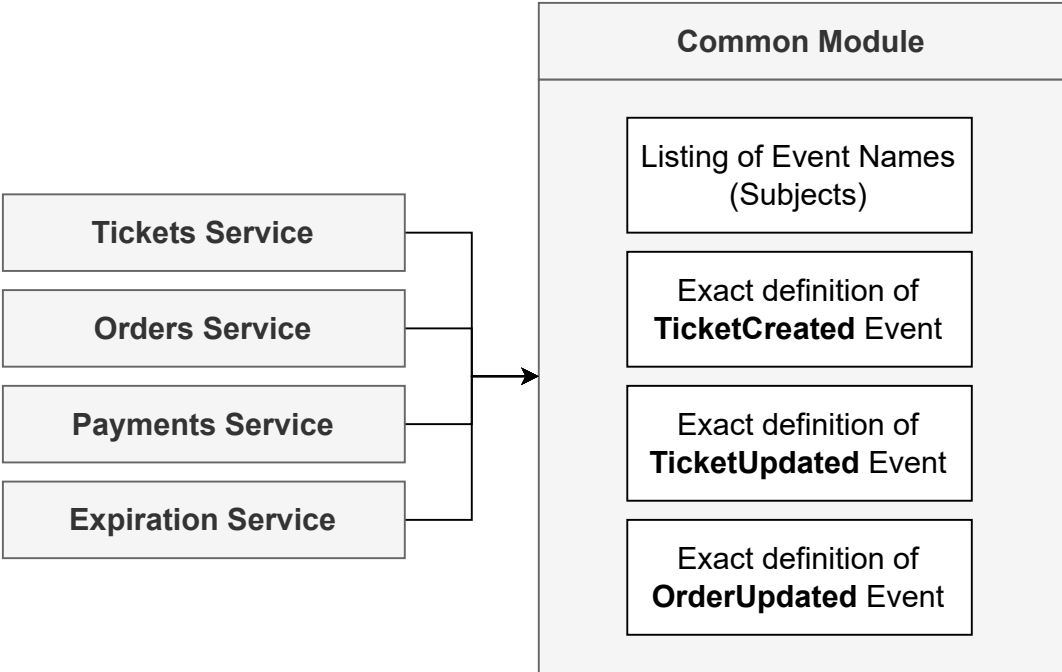


```
graph LR; Publishers[Publishers] --> PDesc[Similar to making a network request. Not a lot to test]; Listeners[Listeners] --> LDesc[Super similar in nature to request handlers! Tons of stuff to test!];
```

Similar to making a network request. Not a lot to test

Listeners

Super similar in nature to request handlers! Tons of stuff to test!



Downside

This only works if all of our servers are written with Typescript!

Tickets Service

Orders Service

Payments Service

Expiration Service



Common Module

Listing of Event Names
(Subjects)

Exact definition of
TicketCreated Event

Exact definition of
TicketUpdated Event

Exact definition of
OrderUpdated Event

Alternatives with Cross Language Support

JSON Schema

Protobuf

Apache Avro