

# Services

**auth**

Everything related to user  
signup/signin/signout

**tickets**

Ticket creation/editing. Knows whether a  
ticket can be updated

**orders**

Order creation/editing

**expiration**

Watches for orders to be created,  
cancels them after 15 minutes

**payments**

Handles credit card payments. Cancels orders if  
payments fails, completes if payment succeeds

## Tickets Service

### Ticket

Prop	Type
title	Title of event this ticket is for
price	Price of the ticket in USD
userId	ID of the user who is selling this ticket
version	Version of this ticket. Increment every time this ticket is changed

**Event  
ticket:created**

**Event  
ticket:updated**

## Orders Service

### Ticket

Prop	Type
title	Title of event this ticket is for
price	Price of the ticket in USD
version	Ensures that we don't process events twice or out of order

### Order

Prop	Type
userId	User who created this order and is trying to buy a ticket
status	Whether the order is expired, paid, or pending
expiresAt	Time at which this order expires (user has 15 mins to pay)
ticketId	ID of the ticket the user is trying to buy

## Tickets Service

### Ticket

Prop	Type
title	Title of event this ticket is for
price	Price of the ticket in USD
userId	ID of the user who is selling this ticket
version	Version of this ticket. Increment every time this ticket is changed

ticket:updated

{ id: 'abc', price: **20**,  
version: **3** }

ticket:updated

{ id: 'abc', price: **30**,  
version: **4** }

ticket:updated

{ id: 'abc', price: **10**,  
version: **2** }

## Orders Service

### Ticket

Prop	Type
title	Title of event this ticket is for
price	Price of the ticket in USD
version	Ensures that we don't process events twice or out of order

### Order

Prop	Type
userId	User who created this order and is trying to buy a ticket
status	Whether the order is expired, paid, or pending
expiresAt	Time at which this order expires (user has 15 mins to pay)
ticketId	ID of the ticket the user is trying to buy



# Orders Service Setup

Duplicate the 'tickets' service

Install dependencies

Build an image out of the orders service

Create a Kubernetes deployment file

Set up file sync options in the skaffold.yaml file

Set up routing rules in the ingress service

## orders

Route	Method	Body	Purpose
/api/orders	GET	-	Retrieve all active orders for the given user making the request
/api/orders/:id	GET	-	Get details about a specific order
/api/orders	POST	{ ticketId: string }	Create an order to purchase the specified ticket
/api/orders/:id	DELETE	-	Cancel the order

We need to somehow associate  
Tickets and Orders together

**Ticket Document**

**Order Document**

There are two primary ways to do  
this with MongoDB/mongoose

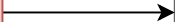
*Option #1 -  
Embedding*

Order	
<i>Prop</i>	<i>Value</i>
userId	'abc'
status	'pending'
expiresAt	1-January
ticket	{ id: 'asdf', price: 12, title: 'concert' }

*Option #1 - Querying is just a bit challenging*

### POST Request

Create order to  
purchase Ticket 'asdf'



## Orders Service

### Order Doc #1

ticket

{ id: '123', price: 12, title: 'concert' }

### Order Doc #2

ticket

{ id: 'XYZ', price: 12, title: 'concert' }

### Order Doc #3

ticket

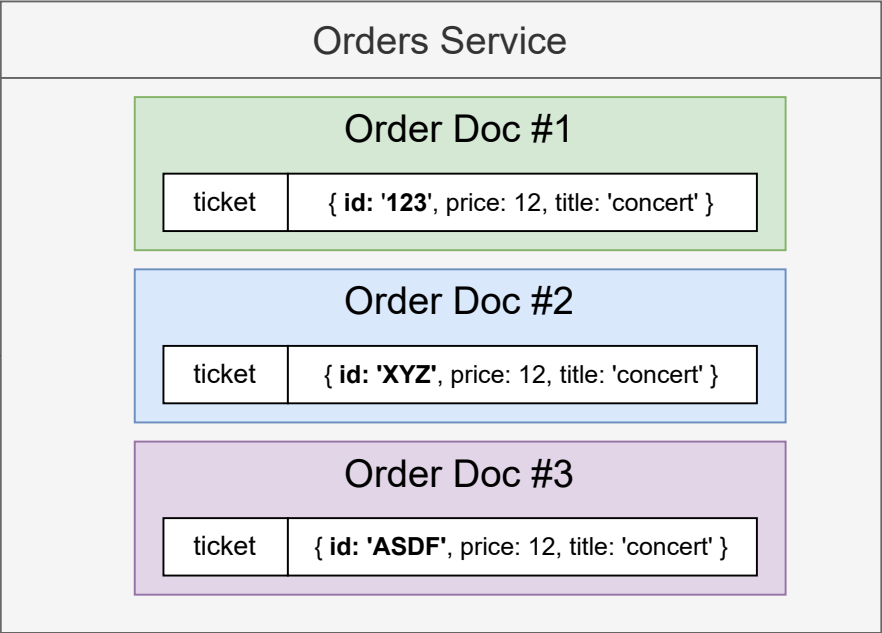
{ id: 'ASDF', price: 12, title: 'concert' }




*Option #1 - Where do we put an unreserved ticket?*

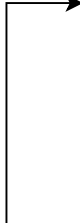
**Event ticket:created**

{ id: 'QQQ', price: 20, title: 'football' }

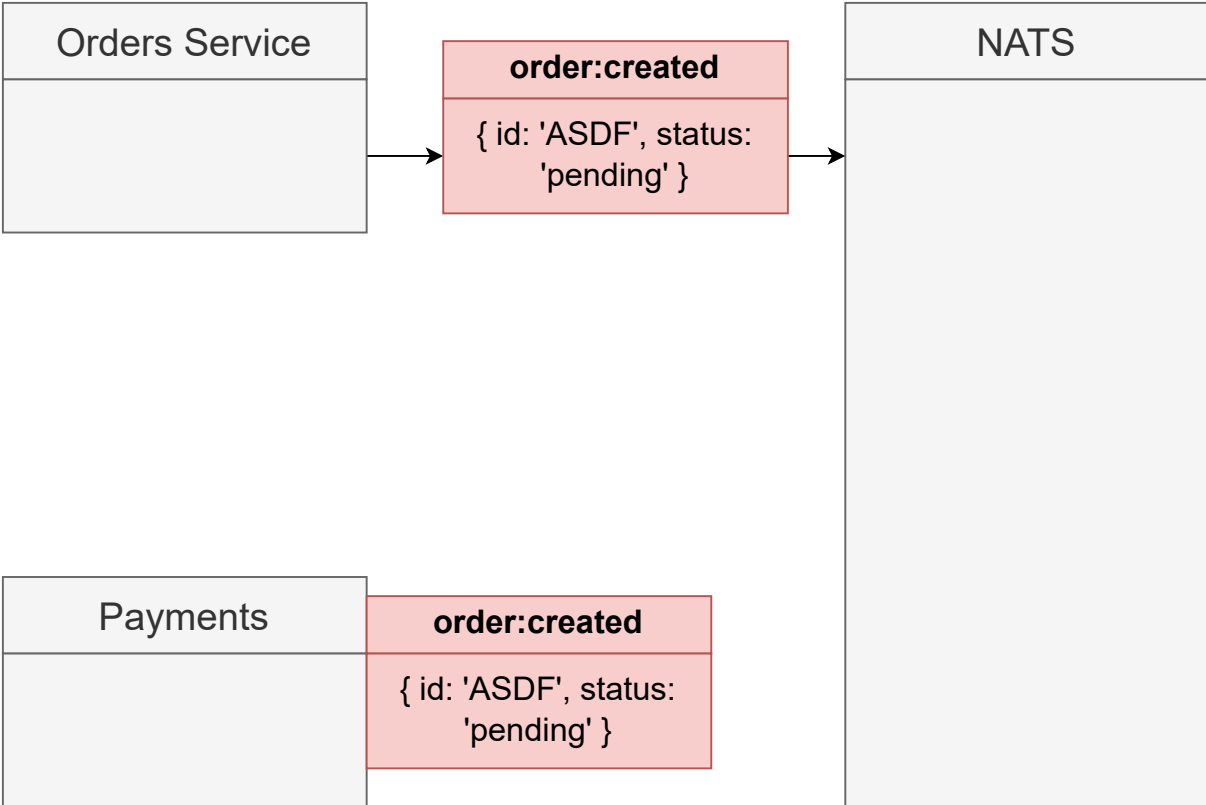


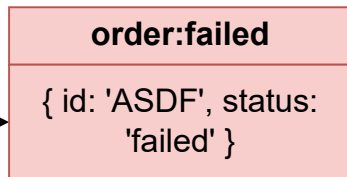
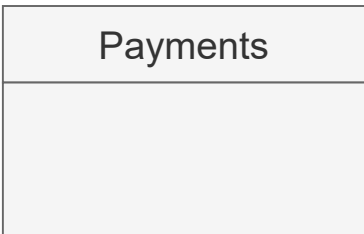
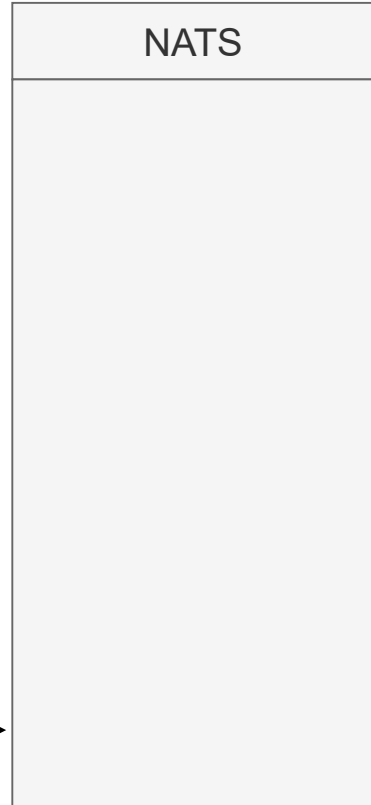
*Option #2 - Mongoose  
Ref/Population Feature*

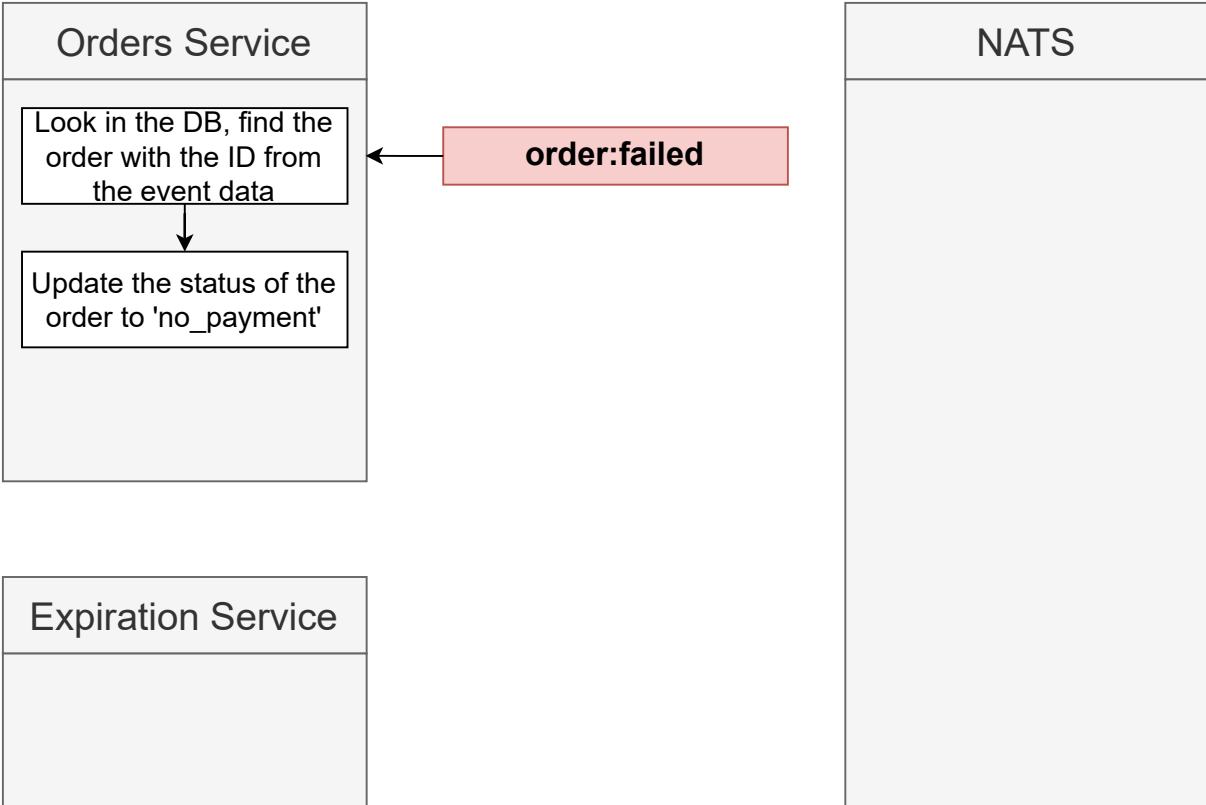
Order Document	
<i>Prop</i>	<i>Value</i>
userId	'abc'
status	'pending'
expiresAt	1-January
ticket	



Ticket Document	
<i>Prop</i>	<i>Value</i>
price	12
title	'concert'







Orders Service

Expiration Service

Payments Service

**Need a *shared and exact* definition of the different statuses an order can have!**

