Metallica: A full-stack microservices based web application

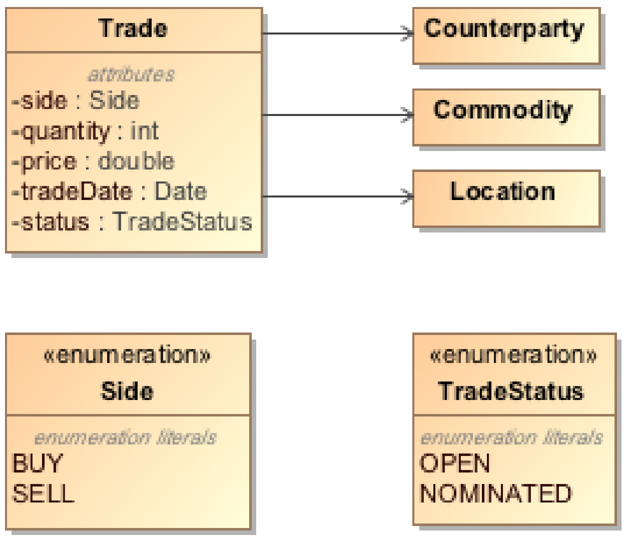
This document outlines an exercise to build a full-stack web application using cutting-edge front-end technologies and microservices. The intent is to implement a reasonably complex business domain that challenges your design and implementation skills. The domain is *physical metals trading,* which involves the following processes:

- Purchase and sale of metals such as aluminium, zinc and copper
- The logistics of moving the material from the place of purchase to the place of the sale

We will provide a high-level architecture for the desired system. You must follow this architecture in your detailed design and implementation. You must also make sure that each component of your system is well tested using automated tests. In addition, your system as a whole should be integration tested.
[This project is named as a tribute to the heavy metal band Metallica, whose fast tempos, instrumentals, and aggressive musicianship placed them as one of the top heavy metal bands in the world.]

Business Domain

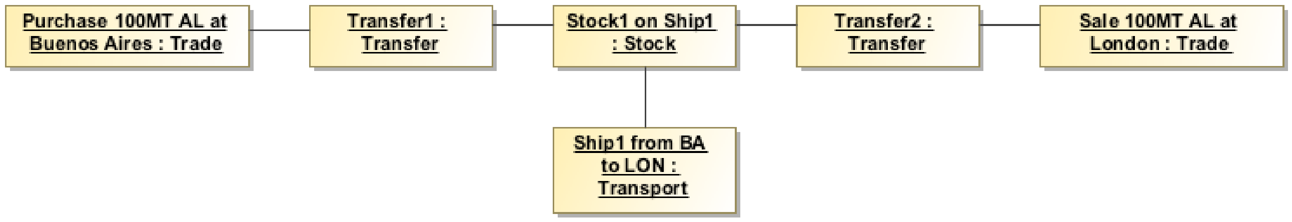
Trades



A *Trade* is a purchase or sale of a commodity from (or to) a counterparty at a specified location, date and price. For example, we could buy 100 MT (metric tons) of aluminium from a counterparty at Buenos Aires on a specified date at $1,860.75/MT and sell it to another counterparty in London on a later date at $2,010.20/MT. In order to meet the sales terms, we would have to transport the material from Buenos Aires to London.
New trades are in *open* status. When they are committed to a transfer, their state changes to *nominated* (more about this below).

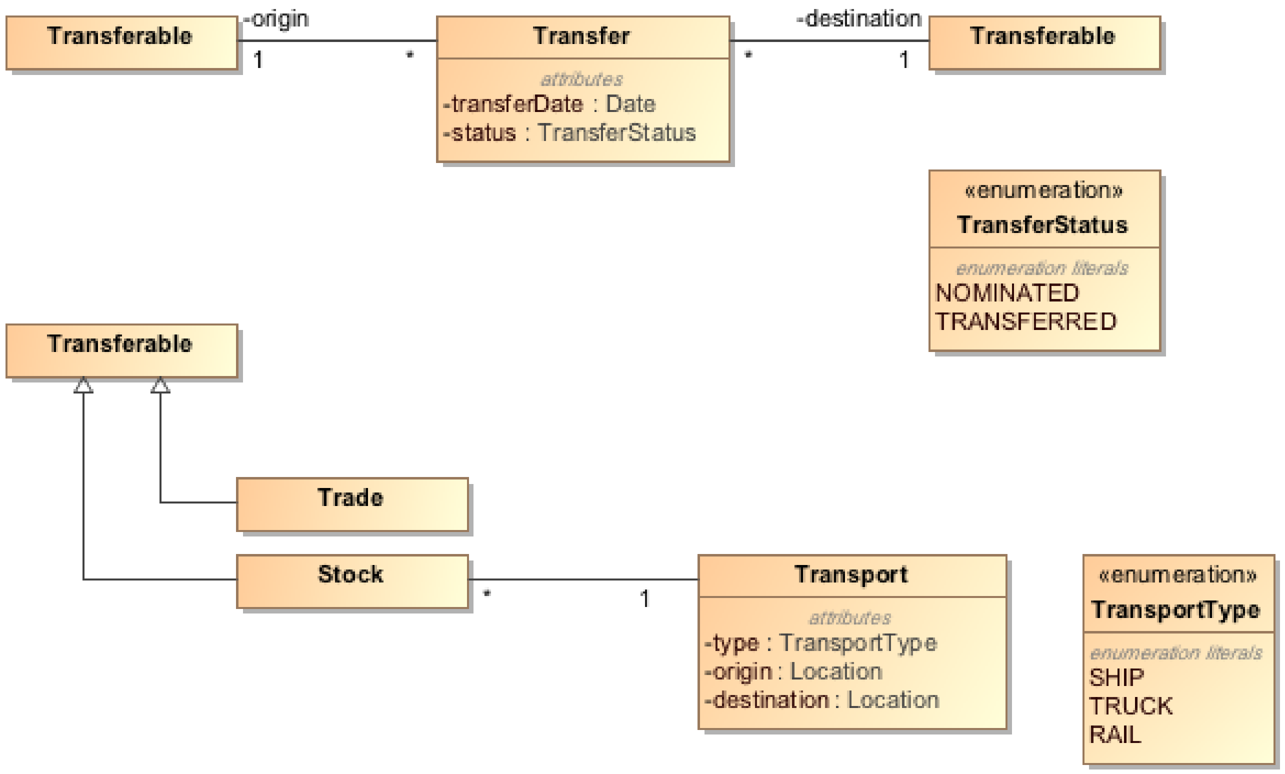
Logistics

Let's use the example above to understand the logistics of moving the material from Buenos Aires to London.


Purchase 100MT AL at Buenos Aires : Trade
Transfer1 : Transfer
Stock1 on Ship1 : Stock
Transfer2 : Transfer
Sale 100MT AL at London : Trade
Ship1 from BA to LON : Transport

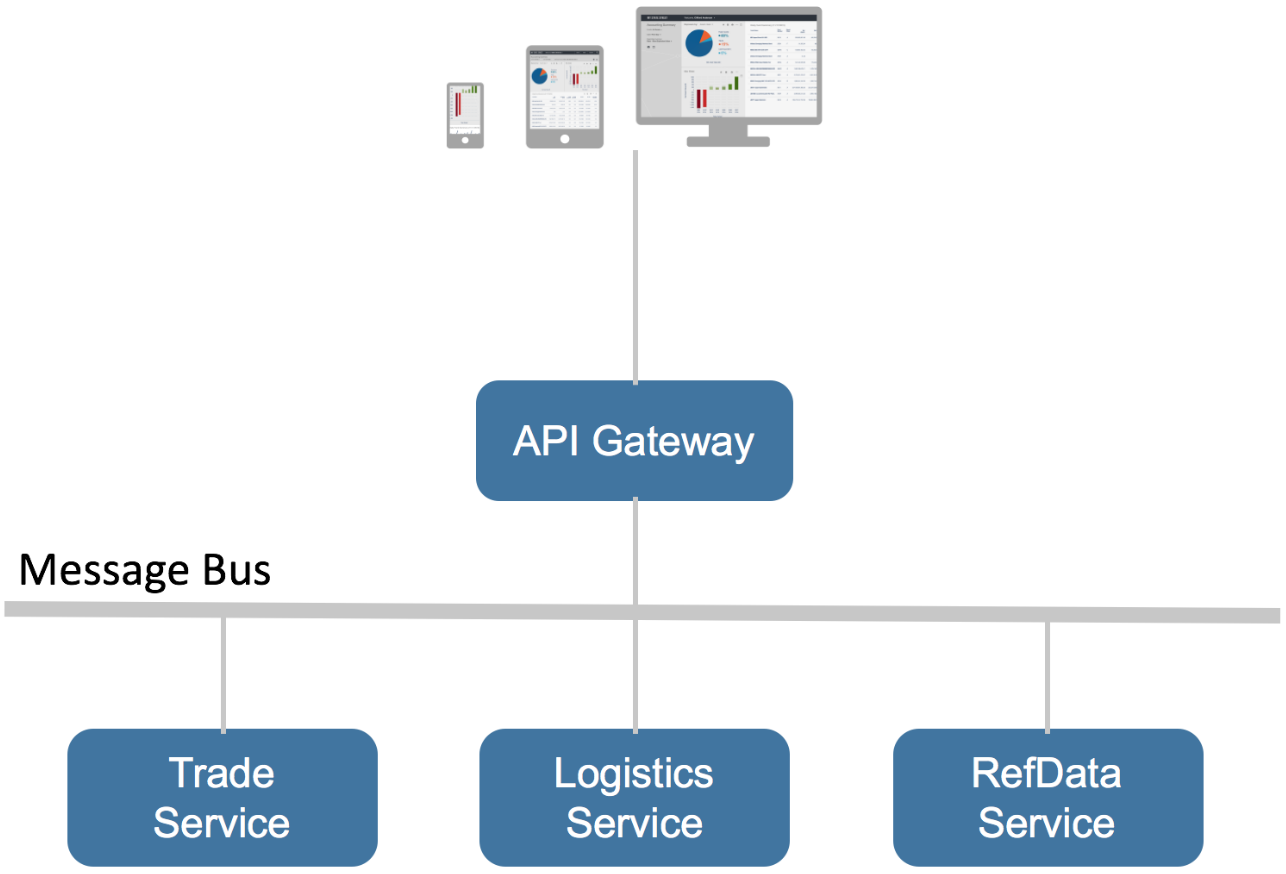
- We *purchase* 100 MT of aluminium at Buenos Aires.
- We *transfer* the material from this purchase on to a ship from Buenos Aires to London. This material is represented as *stock* on the ship.
- When the ship arrives in London, we deliver the material to our customer at that port. This fulfills our obligation of the sale terms.

The domain model below generalizes these concepts:



A *Transfer* involves moving material between two *Transferables*. A transfer has a transfer date and a status (nominated vs. transferred).
A *Transferable* is a *Trade* or a *Stock*. A Stock represents inventory on a *Transport*. Note that a Transport can carry multiple stocks from a single origin to a single destination.

High-Level Architecture
The diagram below illustrates the high-level architecture of Metallica.



The data needs for the front-end will be fulfilled by an API gateway. The gateway uses GraphQL, a query language for APIs and a runtime for fulfilling those queries. While REST APIs typically require loading data from multiple URLs, GraphQL APIs get all the data needed by a front-end screen (or other components) in a single request. GraphQL has the capability to break down a complex query, fetch the results of the sub-queries from various resources and compose the results into a single response to the front-end.
The back-end consists of three micro-services, each responsible for a subset of the business domain (bounded context):

- **TradeService** supports the management of *Trade* instances.
- **LogisticsService** supports *Transports*, *Transferables* and *Transfers*.
- **RefDataService** supports read-only reference entities such as *Counterparties*, *Commodities* and *Locations*. Each reference entity should have an *identifier* that can be used to reference it from other entities and other required attributes. For example, a counterparty may have an *identifier* (e.g. "AAPL") and a *name* (e.g. "Apple, Inc.").

The API gateway and the micro-services communicate with each other using asynchronous messages. This has a number of advantages compared to synchronous messaging, you can read more about it here.
The message bus carries three types of messages:

- **Events** are notifications that inform listeners when something has happened, e.g. a Trade was created. Events never modify state directly.
- **Commands** are actions that modify state. A Command asks a service to do something, e.g. Logistics Service, please create a transfer for this purchase.
- **Request/Responses** ask services for information, e.g. LogisticsService, please give me all transports that originate from Buenos Aires.

Note that each microservice is autonomous and fully encapsulated. It contains its own persistence data store, e.g. a relational, aggregate or graph database. The only way to get data from a microservice is through its messaging API. In other word, a microservice does not expose its data store to the outside world - not even to other microservices. This ensures that microservices are resilient to internal implementation changes.

Exercise

Implement Metallica. You must support the following requirements:

- Allow traders to enter trades as described in the domain model.
- Allow traffic personnel to fulfill trades by moving material from places of purchase to places of sale.

Technology Stack

Use the following technologies in your implementation:

Front-End

- ES6 (ES2015)
- React
- Material UI (use this link only, it's for version 1.x.x which is pre-release but much better than version 0.x.x)
- MobX
- Apollo Client for React (GraphQL client implementation)

GraphQL API Gateway

- ES6 (ES2015)
- Node.js 6.x
- Apollo GraphQL Server
- amqplib (for connecting to RabbitMQ)

Microservices

- Java 8
- Spring Boot
- Spring AMQP (for connecting to RabbitMQ)
- Spring Data (for connecting to MongoDB)

Messaging Infrastructure

- RabbitMQ

Database

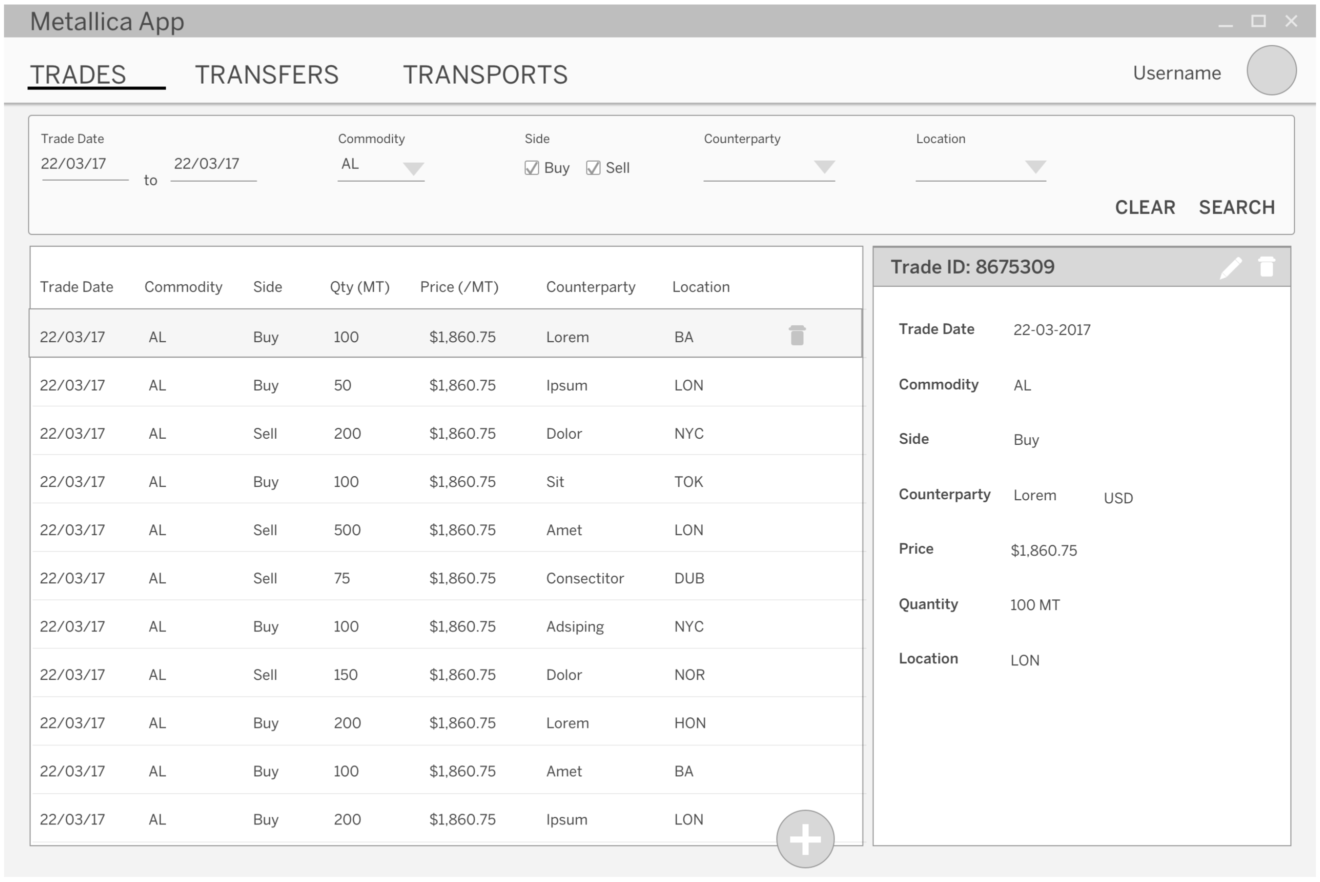
- MongoDB

Front-End Design

The front-end design for Metallica consists of three screens: Trades, Transfers and Transports.

Trades Screen

This screen is used to search, create, edit, view and delete trades.

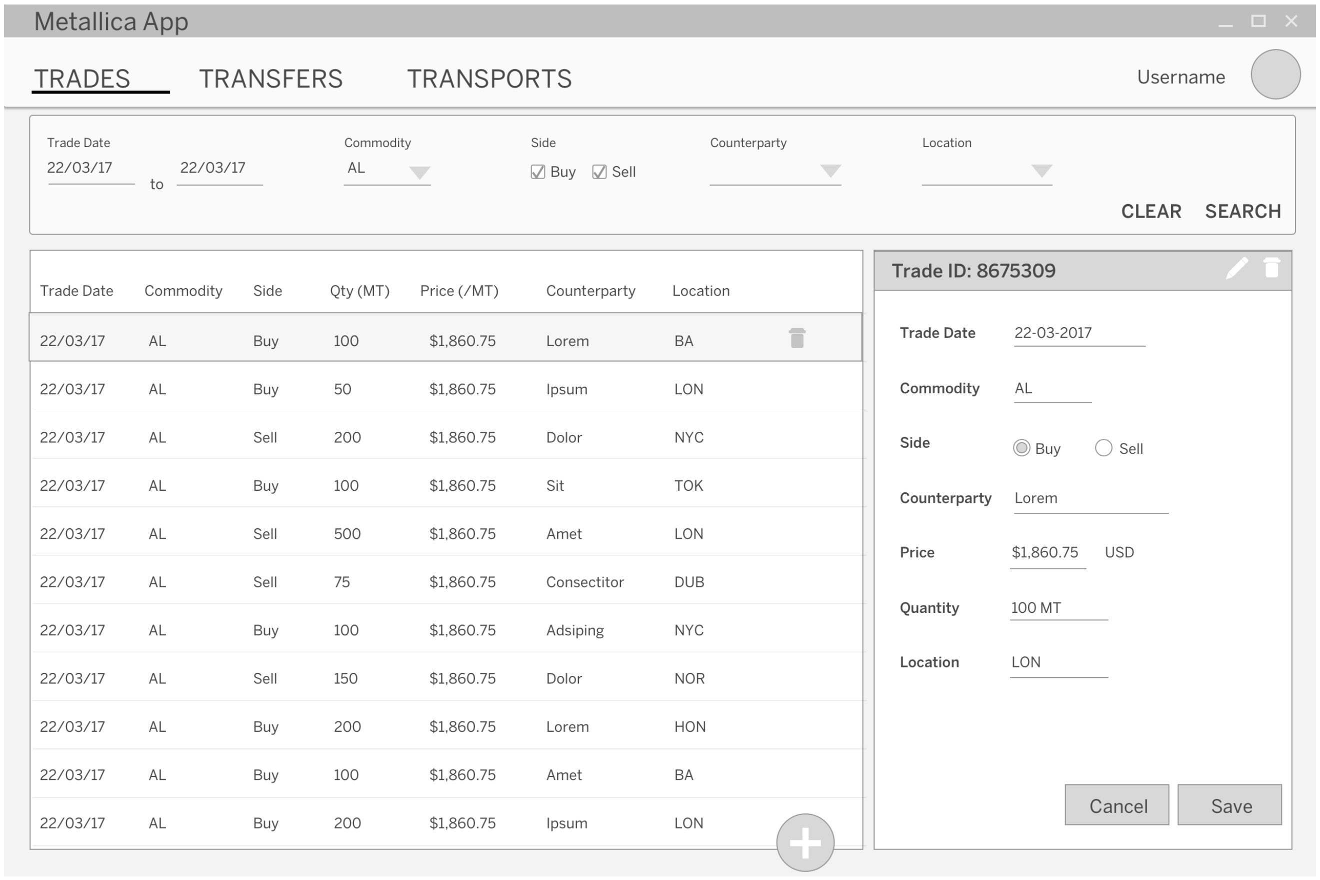


Trade Date	Commodity	Side	Qty (MT)	Price (/MT)	Counterparty	Location
22/03/17	AL	Buy	100	$1,860.75	Lorem	BA
22/03/17	AL	Buy	50	$1,860.75	Ipsum	LON
22/03/17	AL	Sell	200	$1,860.75	Dolor	NYC
22/03/17	AL	Buy	100	$1,860.75	Sit	TOK
22/03/17	AL	Sell	500	$1,860.75	Amet	LON
22/03/17	AL	Sell	75	$1,860.75	Consectitor	DUB
22/03/17	AL	Buy	100	$1,860.75	Adsiping	NYC
22/03/17	AL	Sell	150	$1,860.75	Dolor	NOR
22/03/17	AL	Buy	200	$1,860.75	Lorem	HON
22/03/17	AL	Buy	100	$1,860.75	Amet	BA
22/03/17	AL	Buy	200	$1,860.75	Ipsum	LON

Trade ID: 8675309

Trade Date	22-03-2017
Commodity	AL
Side	Buy
Counterparty	Lorem USD
Price	$1,860.75
Quantity	100 MT
Location	LON

Below is an example of a trade being edited:



Trade Date 22/03/17 to 22/03/17 | Commodity AL | Side [✓ Buy] [✓ Sell] | Counterparty | Location

CLEAR SEARCH

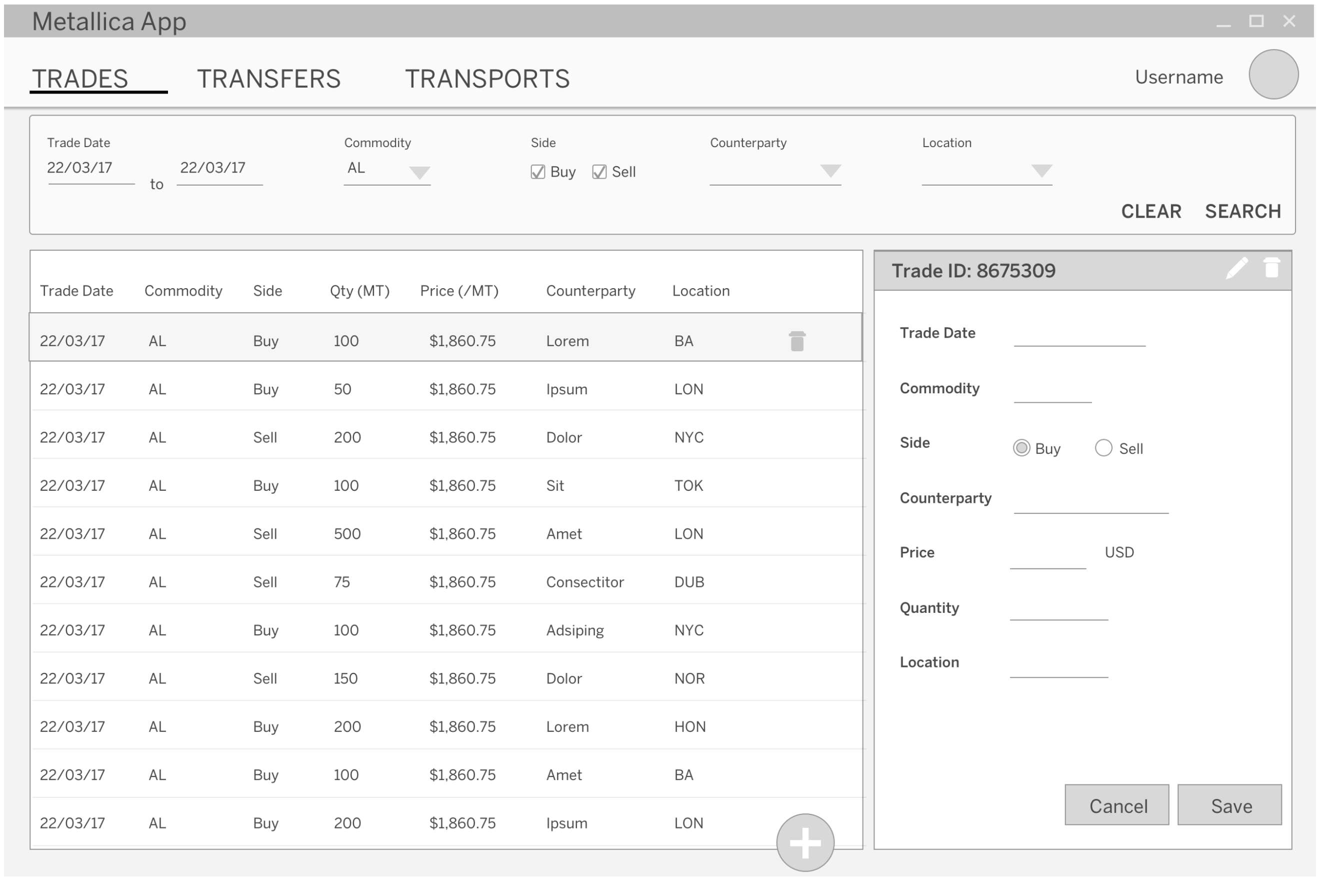
Trade Date	Commodity	Side	Qty (MT)	Price (/MT)	Counterparty	Location
22/03/17	AL	Buy	100	$1,860.75	Lorem	BA
22/03/17	AL	Buy	50	$1,860.75	Ipsum	LON
22/03/17	AL	Sell	200	$1,860.75	Dolor	NYC
22/03/17	AL	Buy	100	$1,860.75	Sit	TOK
22/03/17	AL	Sell	500	$1,860.75	Amet	LON
22/03/17	AL	Sell	75	$1,860.75	Consectitor	DUB
22/03/17	AL	Buy	100	$1,860.75	Adsiping	NYC
22/03/17	AL	Sell	150	$1,860.75	Dolor	NOR
22/03/17	AL	Buy	200	$1,860.75	Lorem	HON
22/03/17	AL	Buy	100	$1,860.75	Amet	BA
22/03/17	AL	Buy	200	$1,860.75	Ipsum	LON

Trade ID: 8675309

Trade Date	22-03-2017
Commodity	AL
Side	[✓ Buy] Sell
Counterparty	Lorem
Price	$1,860.75 USD
Quantity	100 MT
Location	LON

Cancel Save

Finally, here's an example of a new trade being created:



Trade Date 22/03/17 to 22/03/17 | Commodity AL | Side ☑ Buy ☑ Sell | Counterparty | Location

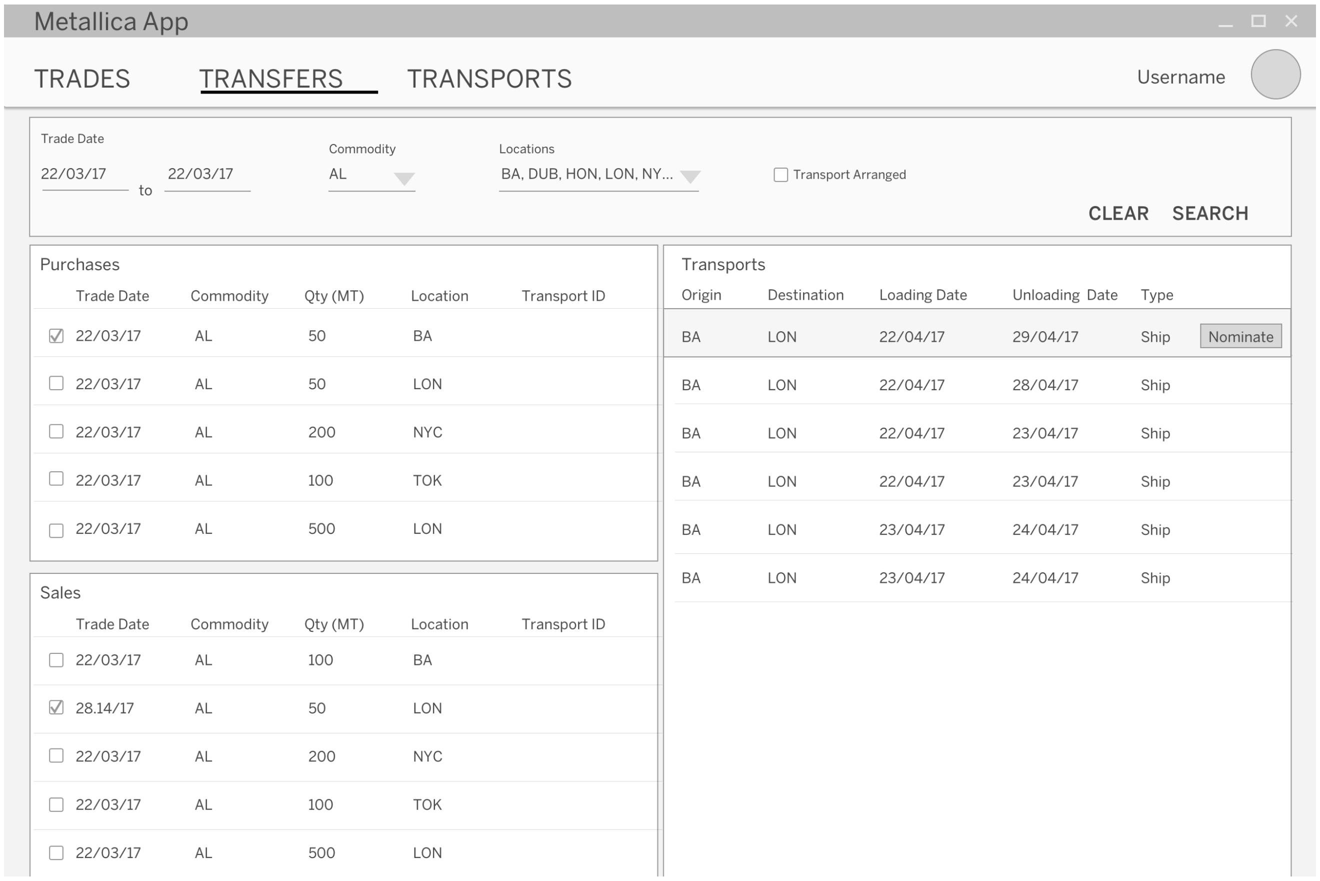
Trade Date	Commodity	Side	Qty (MT)	Price (/MT)	Counterparty	Location
22/03/17	AL	Buy	100	$1,860.75	Lorem	BA
22/03/17	AL	Buy	50	$1,860.75	Ipsum	LON
22/03/17	AL	Sell	200	$1,860.75	Dolor	NYC
22/03/17	AL	Buy	100	$1,860.75	Sit	TOK
22/03/17	AL	Sell	500	$1,860.75	Amet	LON
22/03/17	AL	Sell	75	$1,860.75	Consectitor	DUB
22/03/17	AL	Buy	100	$1,860.75	Adsiping	NYC
22/03/17	AL	Sell	150	$1,860.75	Dolor	NOR
22/03/17	AL	Buy	200	$1,860.75	Lorem	HON
22/03/17	AL	Buy	100	$1,860.75	Amet	BA
22/03/17	AL	Buy	200	$1,860.75	Ipsum	LON

Trade ID: 8675309

Trade Date	
Commodity	
Side	[✓ Buy] Sell
Counterparty	
Price	USD
Quantity	
Location	

Transfers Screen

This screen is used for arranging transports for matching purchases and sales. As discussed in the domain model, purchases must be loaded (or *transferred*) on to a transport and unloaded at the the location of the sale. Imagine a scenario where we are approaching a new month and the traffic person (person in-charge of logistics) needs to arrange transports for that month. They can first search for all trades with trade dates in that month. They can then choose matching purchases and sales. As they make these selections, the transports on the right get filtered to those available between the selected origin and destination with favorable loading and unloading dates. The user can then *nominate* a transport for the selected trades.



Purchases

	Trade Date	Commodity	Qty (MT)	Location	Transport ID
✓	22/03/17	AL	50	BA	
☐	22/03/17	AL	50	LON	
☐	22/03/17	AL	200	NYC	
☐	22/03/17	AL	100	TOK	
☐	22/03/17	AL	500	LON	

Sales

	Trade Date	Commodity	Qty (MT)	Location	Transport ID
☐	22/03/17	AL	100	BA	
✓	28.14/17	AL	50	LON	
☐	22/03/17	AL	200	NYC	
☐	22/03/17	AL	100	TOK	
☐	22/03/17	AL	500	LON	

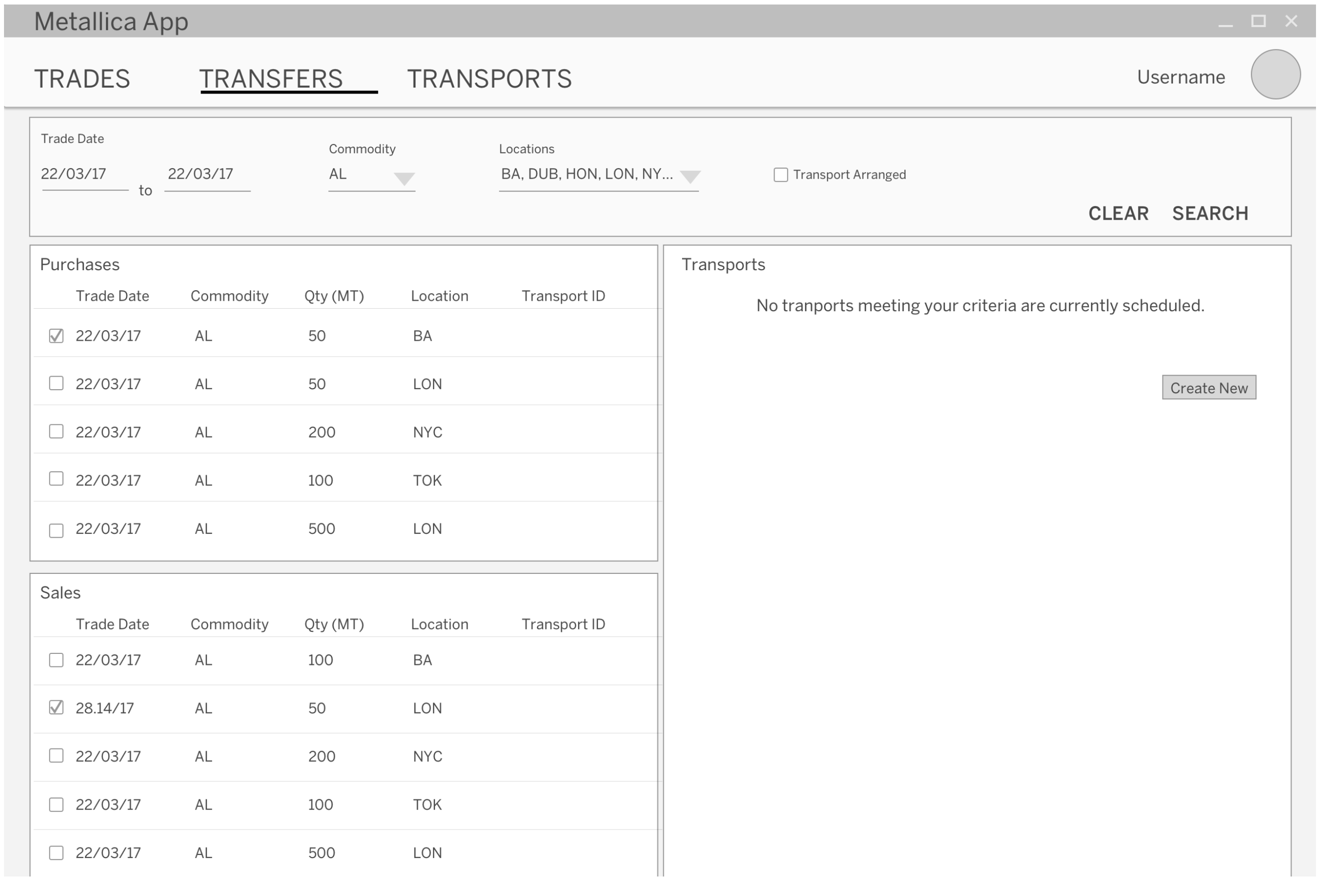
Transports

Origin	Destination	Loading Date	Unloading Date	Type	
BA	LON	22/04/17	29/04/17	Ship	Nominate
BA	LON	22/04/17	28/04/17	Ship	
BA	LON	22/04/17	23/04/17	Ship	
BA	LON	22/04/17	23/04/17	Ship	
BA	LON	23/04/17	24/04/17	Ship	
BA	LON	23/04/17	24/04/17	Ship	

When a purchase and a sale are nominated to a transport, the system performs the following actions:

- A stock is created on the transport.
- A load transfer is created between the purchase and the stock. The purchase is set to the *nominated* status.
- An unload transfer is created between the stock and the sale. The sale is set to the *nominated* status.

Note that it is possible that after selecting purchases and sales, there is no matching transport available. In this case the RHS will be empty and the user will be able to create a new transport:



No tranports meeting your criteria are currently scheduled.

Purchases

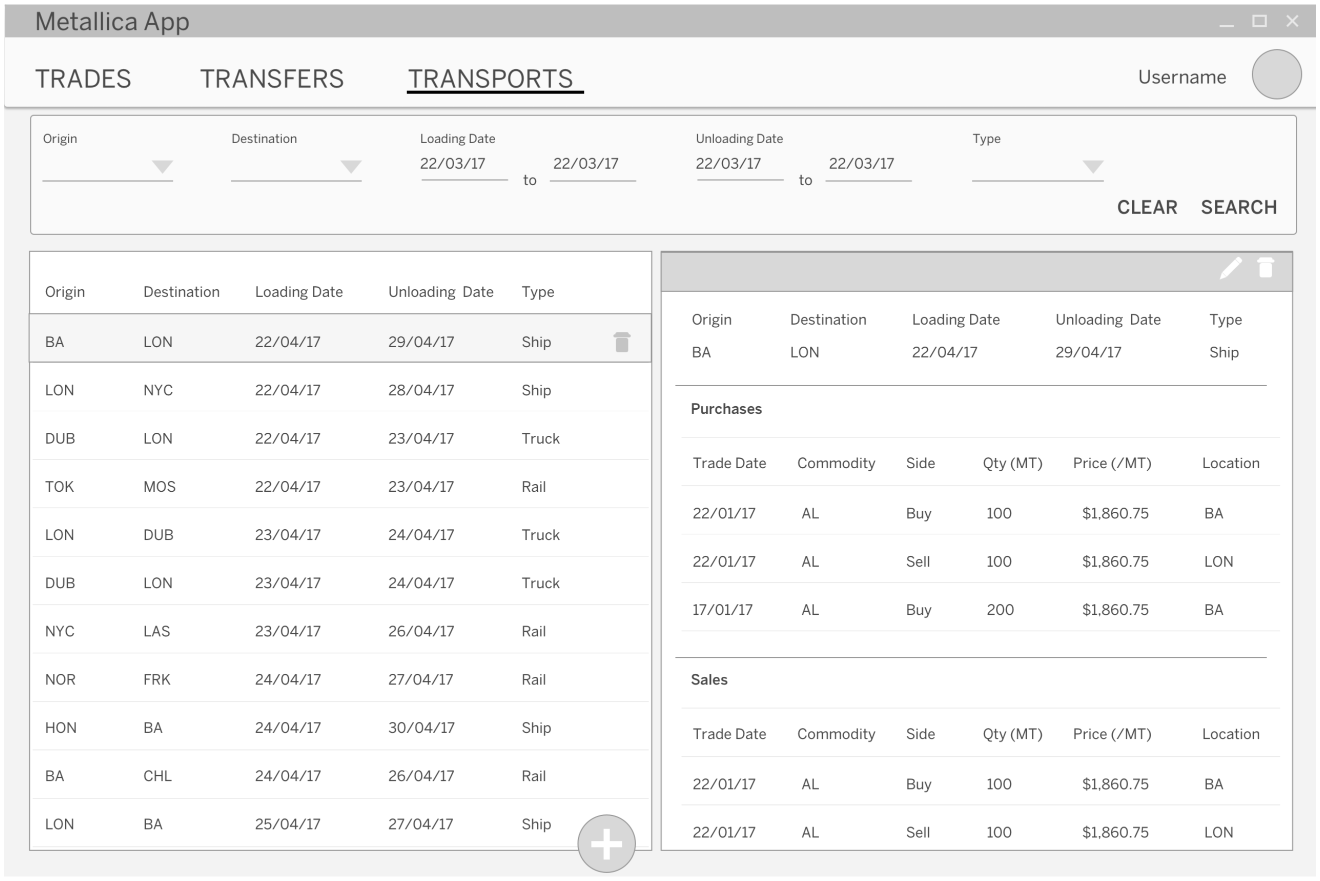
	Trade Date	Commodity	Qty (MT)	Location	Transport ID
✓	22/03/17	AL	50	BA	
☐	22/03/17	AL	50	LON	
☐	22/03/17	AL	200	NYC	
☐	22/03/17	AL	100	TOK	
☐	22/03/17	AL	500	LON	

Sales

	Trade Date	Commodity	Qty (MT)	Location	Transport ID
☐	22/03/17	AL	100	BA	
✓	28.14/17	AL	50	LON	
☐	22/03/17	AL	200	NYC	
☐	22/03/17	AL	100	TOK	
☐	22/03/17	AL	500	LON	

Transports Screen

This screen is used to search, create, edit, view and delete transports. This screen is mostly used by logistics people when they need to load or unload a transport. When a transport is selected the RHS shows the material that should be loaded/unloaded.



Origin	Destination	Loading Date	Unloading Date	Type
BA	LON	22/04/17	29/04/17	Ship
LON	NYC	22/04/17	28/04/17	Ship
DUB	LON	22/04/17	23/04/17	Truck
TOK	MOS	22/04/17	23/04/17	Rail
LON	DUB	23/04/17	24/04/17	Truck
DUB	LON	23/04/17	24/04/17	Truck
NYC	LAS	23/04/17	26/04/17	Rail
NOR	FRK	24/04/17	27/04/17	Rail
HON	BA	24/04/17	30/04/17	Ship
BA	CHL	24/04/17	26/04/17	Rail
LON	BA	25/04/17	27/04/17	Ship

Origin	Destination	Loading Date	Unloading Date	Type
BA	LON	22/04/17	29/04/17	Ship

Purchases

Trade Date	Commodity	Side	Qty (MT)	Price (/MT)	Location
22/01/17	AL	Buy	100	$1,860.75	BA
22/01/17	AL	Sell	100	$1,860.75	LON
17/01/17	AL	Buy	200	$1,860.75	BA

Sales

Trade Date	Commodity	Side	Qty (MT)	Price (/MT)	Location
22/01/17	AL	Buy	100	$1,860.75	BA
22/01/17	AL	Sell	100	$1,860.75	LON

Testing Approach

Be sure to define *Behavioral Unit Tests* for the front-end and each microservice. If you don't know what that means, watch this video:
Ian Cooper: TDD, where did it all go wrong
The most important take-away from this should be the understanding that ***our testing strategy leverages the Ports and Adapters Architecture to allow the units under test to be tested directly, but in terms of the behavior we are actually interested in***. Once you understand this concept, you will be well equipped to test your system.

Happy Implementation!