

Our micro services architecture

A micro-service Rest API that manages stocks, stock orders and user's stock holdings



A micro-service Rest API that manages user accounts and authorizations



A micro-service Rest API that manages user's balance in currencies (not stocks)



Our REST APIs methods list (HTTP endpoints)

UserService API METHODS ENDPOINTS

POST	/users/register	Registers a user in the system
POST	/users/login	Logins a user in the system
		Returns a list of users in the system
GET*	/users	Add query (type="investor" type="broker") to limit to investors or brokers.
GET*	/users/:userId	Returns a user's profile by id.
PUT*	/users/:userId	Updates user's profile details.
GET*	/authorizations	Returns a user's authorizations list
GET*	/authorizations/:authorizationId	Returns authorization's details by id
POST*	/authorizations	Create an authorization Request body needs to have a type (Capital authorization or StockHolding authorization)
PUT*	/authorizations/:authorizationId/revoke	User can revoke an authorization access

WalletService API METHODS ENDPOINTS

GET*	/balance	Returns a user's balance
PUT**	/balance	Update user's balance called when selling/buying stocks from the StockService
GET*	/transactions	Returns a user's transaction list
GET*	/transactions/:transactionId	Returns a user's transaction by id
POST*	/transactions	Creates a new transaction. Request body needs to have a type (type = "deposit" type = "withdraw")

StockService API METHODS ENDPOINTS

GET	/stocks	Returns a list of stocks in the system
GET	/stocks/:stockId	Returns a stock by its id
GET*	/stockholdings	Returns a user's stockholdings lists
GET*	/stockholdings/:stockholdingId	Returns a specific stockholding Id.
PUT*	/stockholdings/:stockholdingId	Update committedAmount when authorizing brokers
GET*	/orders	Returns a user's order list
GET*	/orders/:orderId	Return a user's order by id
POST*	/orders	Creates an order to buy/sell stocks If the request is done by a broker, the request body should include "authorizationId", otherwise leave null. Only investors can create an automated order (if the authorizationId exists, the request will return invalid).
PUT*	/orders/:orderId	Update an order's status. If it's not completed yet, you can set the status to cancelled.

* All requests with (*) are authenticated requests. You need to be logged in to execute them. The user's id (which is the user's authentication in our simplified case) will be used in an HTTP authorization Bearer token scheme inside a JWT (JSON Web token) which will be included on the http authentication header of every HTTP call towards our APIs methods that require authentication.

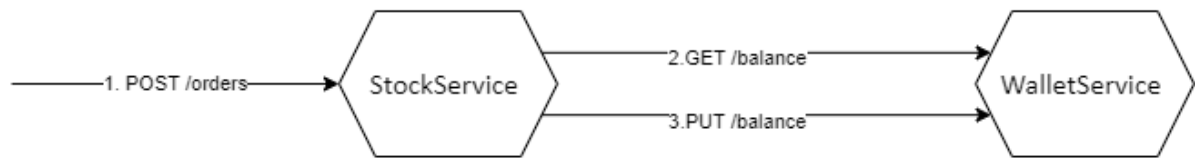
** System requests. Not accessible from public users.

This architecture of microservices allows us to decouple different business logic into logical components that can be deployed separately, while also achieving **low coupling**.

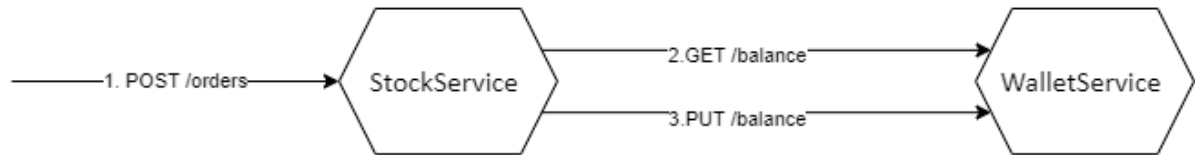
In this architecture, only the POST /orders has high coupling, due to the fact that it needs to verify the authorization details when a request is received from a broker as well as update the states on both microservices. Getting a user's balance also relies on different microservices, because the user's profile is fetched from the UserService while the balance is retrieved from the WalletService.

In the next page is the image that explains the communication between microservices. During development we may end up making changes.

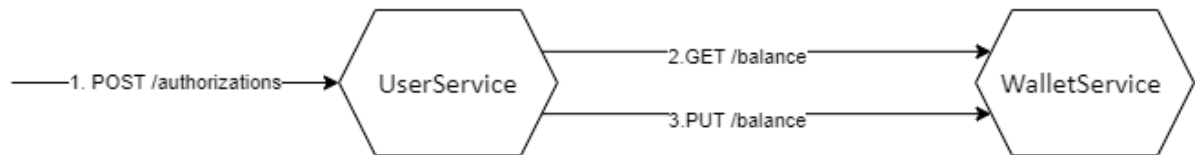
Buy Stock



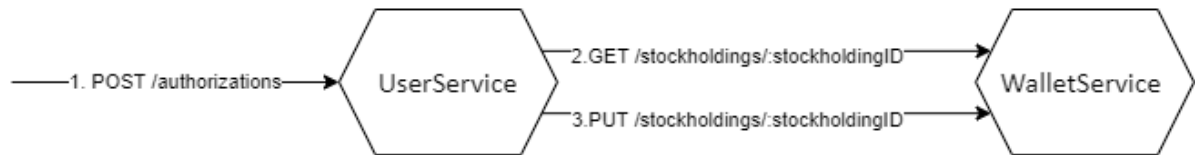
Sell Stock



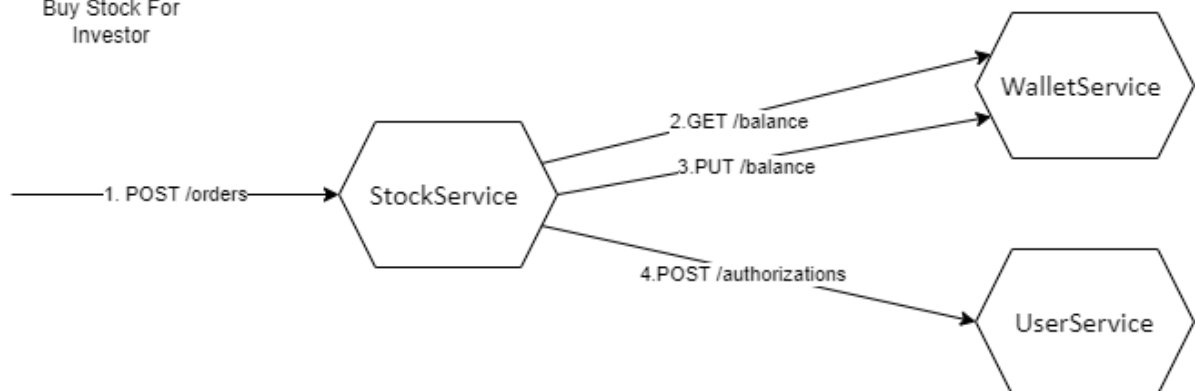
Give Capital Authorization



Give Stock Authorization



Buy Stock For Investor



Sell Stock For Investor

