# CSGO BANK

## Introduction

CSGO is an FPS game that has a lot of items you can get with drops, unboxing, or trading.

The items don’t give you any form of advantage in the game, it is only cosmetic and like a status for having the most “good looking” and expensive items. For example, an AK-47 skin is still the default AK-47 mechanically speaking, but the skin can make it look in a ton of different ways. Items (skins) can be for weapons, knifes, gloves, stickers to put on the weapons and so on. Some skins are very rare and can have an insane price in real fiat currency (pun intended).

So, our project CSGO BANK will be a secure vault, like an inventory if you will, to store your items and operate transactions of these items to other users or sites.

## Planning

### User Stories

As a user i want to…

* Log in to vault.
* Deposit skins to vault.
* Withdraw skins from vault.
* Transfer skins.
  + To other users in same bank.
  + To other users in different bank.
  + To other sites.
* Lock skins.
  + For an amount of time to be unable to transfer.
* Display vault
  + Display items
  + Display info on skins.
  + Check price of skin.
  + Display total value of vault.
* Click on skin to display price history.
* Display transaction history on account.
* Sort skins
  + On skin type
  + On price

### Prioritized user Story

The things I want to prioritize first are as follows.

1. Log in.

2. Display vault.

3. Deposit /withdraw.

## Process

* First, we made a monolith with Vault and Item class with the very basic red thread from frontend to Database. This way we had a “base” structure of our “main” Service.
* Then we wanted to focus on the user story “As a user I want to transfer a skin from my account to another” and wanted to implement RabbitMQ in this process.

### RabbitMQ

* Started checking out how to implement and use RabbitMQ.
* Wished to use RabbitMQ as a broker between vault and transaction.
* Vault asks transaction if transaction is valid, transaction returns answer that it is valid or not.
* -> Vault sends validation message to queue.

-> Transaction subscribe and gets the massage.

-> Transaction sends validation response message to queue.

-> Vault subscribes to this queue (check describing diagrams under).

* After a lot of discussion and trying, like for example, how to differentiate between different transactions if we have more than one instance of “Vault”, maybe an ID in the message? All in all, we found a lot of issues with this and that this way of “two way” communication between services that needs to be synchronized is not suited for using RabbitMQ. Vault needs a response that the transaction is valid before it can do anything else.

**What to do**

* We concluded with two steps:
  + Make a service that we think make more sense to use with RabbitMQ. (async)
    - We know that we want some type of “transaction history” and we believe this is a good candidate for a service to use with RabbitMQ.
    - -> Vault publishes a message that a transaction has been done.

-> RabbitMQ puts it in a queue.

-> “Transaction History” subscribes to that queue and update DB accordingly.

* + Then make a direct communication between Vault and Transaction validator. (sync)

**Diagrams describing the planning of implementing RabbitMQ**.

How we first thought to implement the Message broker.

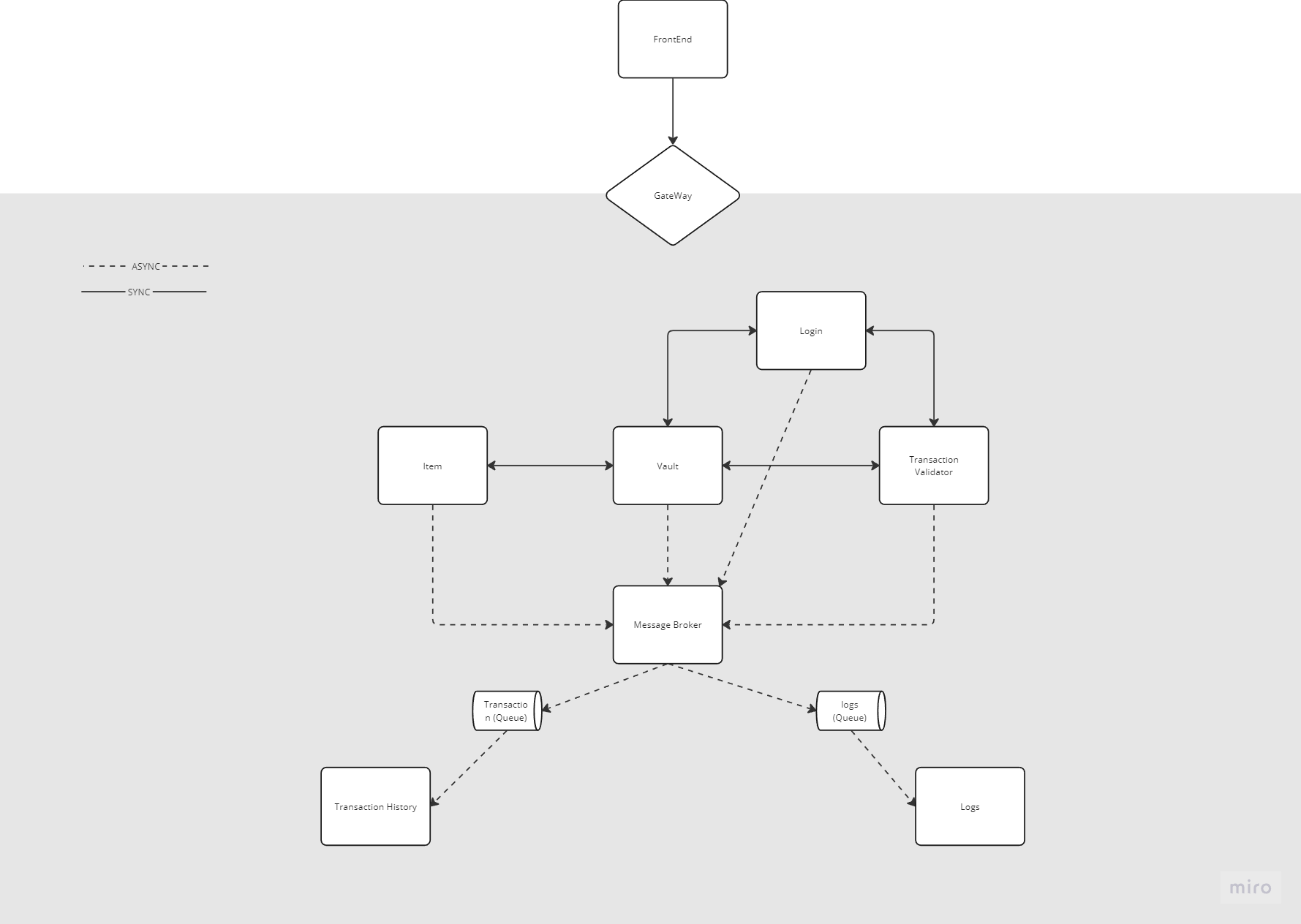
Et bilde som inneholder diagram, plan, Teknisk tegning, skjematisk

Automatisk generert beskrivelse

As a result of research and discussion this is how we decided to set it up the system with message broker and direct communication.

Et bilde som inneholder diagram, Teknisk tegning, plan, line

Automatisk generert beskrivelse

**Diagram with first draft on how we plan to structure our system.**