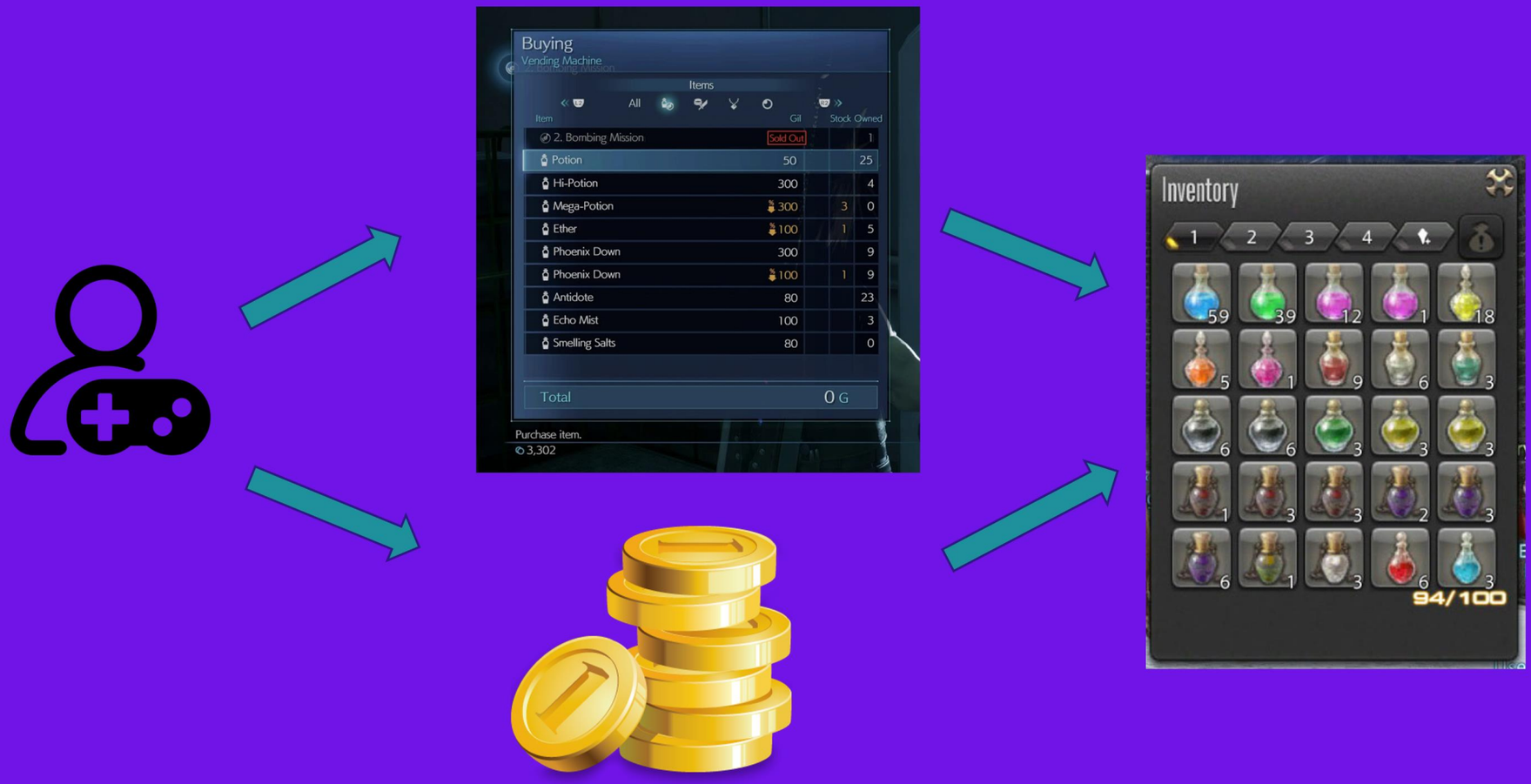
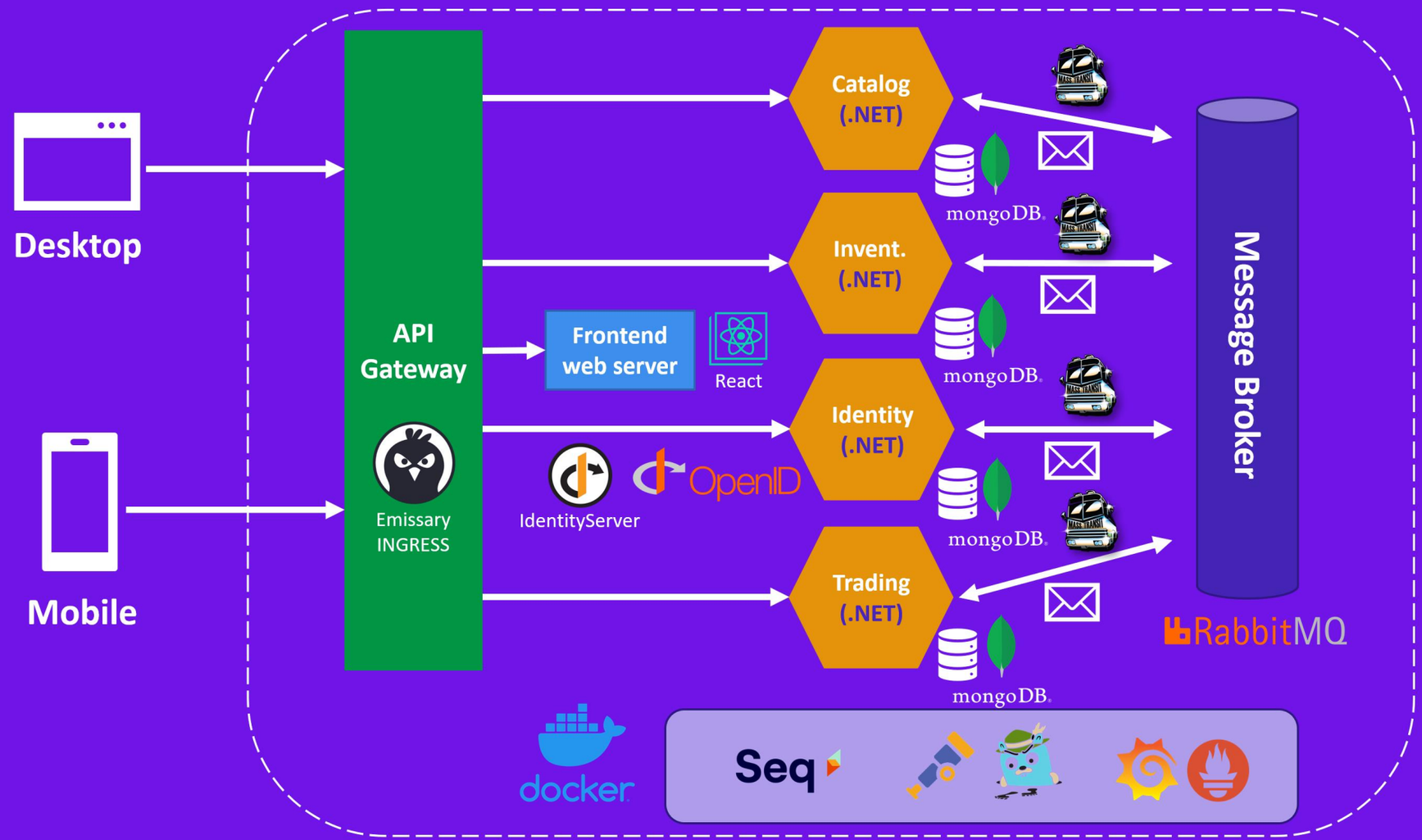
The client application



Play Economy system architecture



Monolith pros and cons

PROS	CONS
Convenient for new projects	Easily gets too complex to understand
Tools mostly focused on them	Merging code can be challenging
Great code reuse	Slows down IDEs
Easier to run locally	Long build times
Easier to debug and troubleshoot	Slow and infrequent deployments
One thing to build	Long testing and stabilization periods
One thing to deploy	Rolling back is all or nothing
One thing to test end to end	No isolation between modules
One thing to scale	Can be hard to scale

Microservices pros and cons

PROS

- Small, easier to understand code base
- Quicker to build
- Independent, faster deployments and rollbacks
- Independently scalable
- Much better isolation from failures
- Designed for continuous delivery
- Easier to adopt new, varied tech
- Grants autonomy to teams and lets them work in parallel

CONS

- Not easy to find the right set of services
- Adds the complexity of distributed systems
- Shared code moves to separate libraries
- No good tooling for distributed apps
- Releasing features across services is hard
- Hard to troubleshoot issues across services
- Can’t use transactions across services
- Raises the required skillset for the team

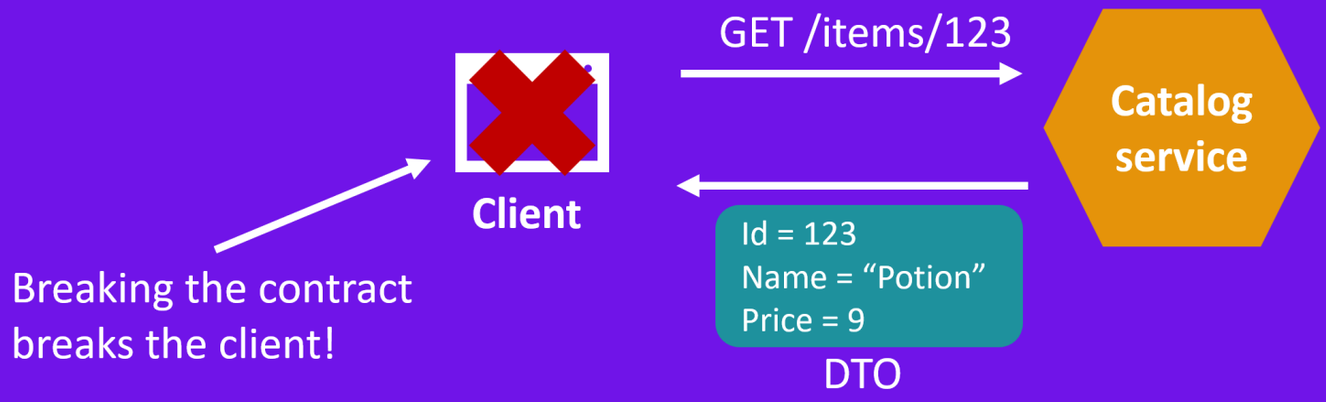
Catalog REST API

The REST API defines the operations exposed by the microservice

Operation	Description
GET /items	Retrieves all items
GET /items/{id}	Retrieves the specified item
POST /items	Creates an item
PUT /items/{id}	Updates the specified item
DELETE /items/{id}	Deletes the specified item

Data Transfer Objects (DTOs)

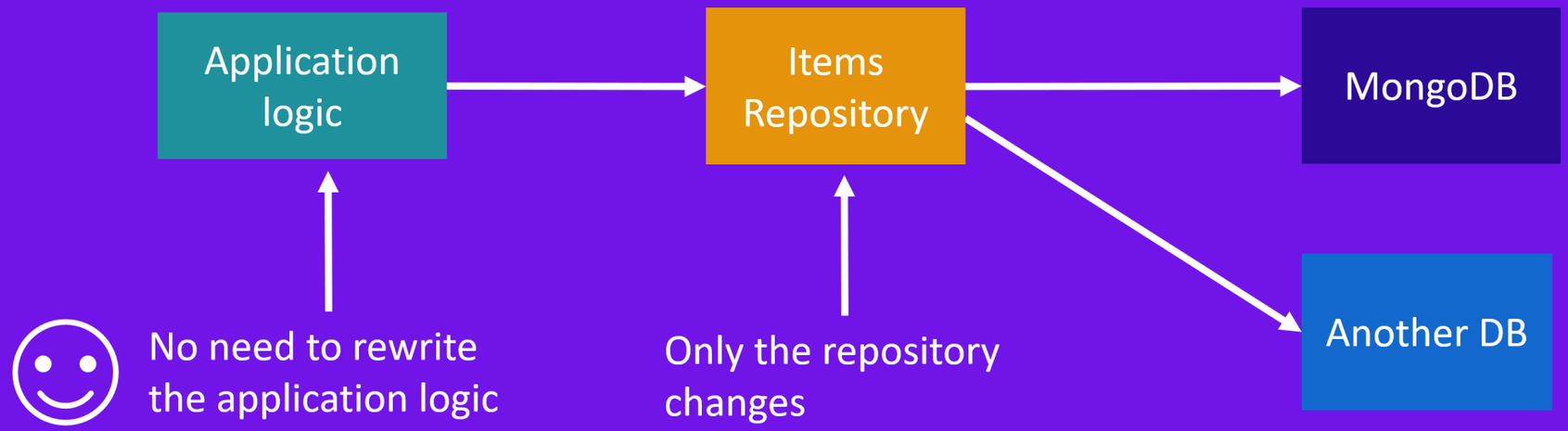
A Data Transfer Object (DTO) is an object that carries data between processes.



The DTO represents the contract between the microservice API and the client

What is a repository?

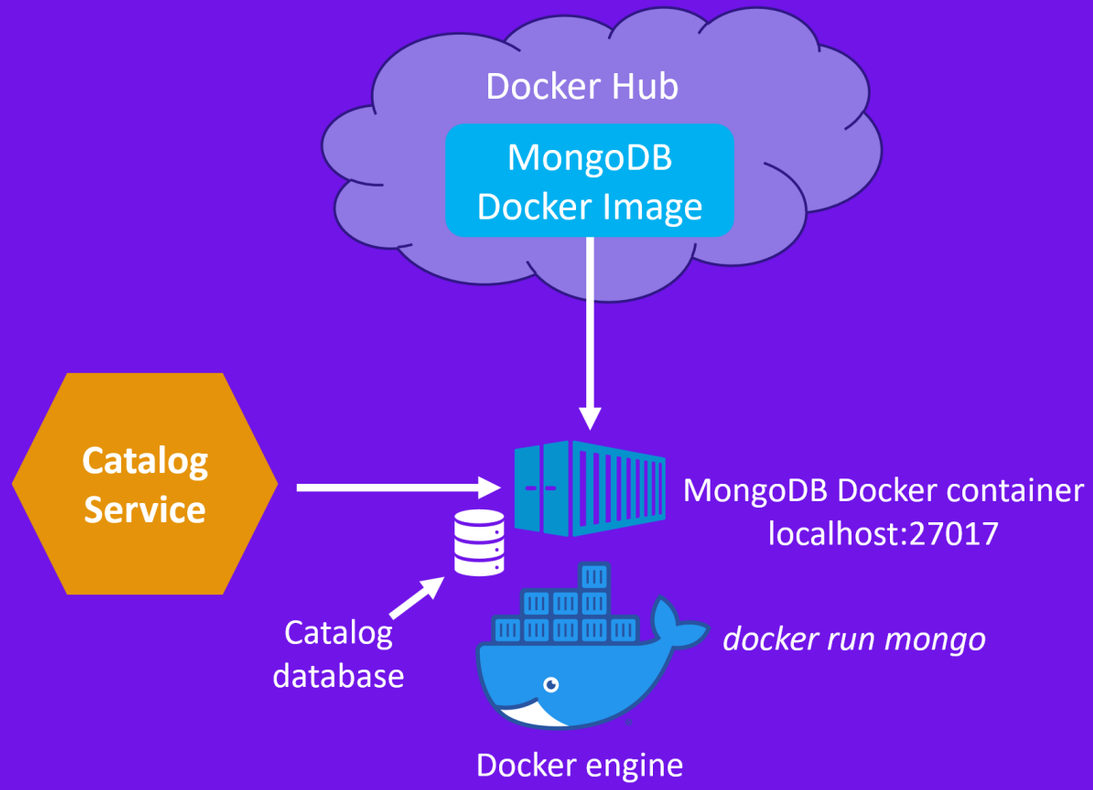
A repository is an abstraction between the data layer and the business layer of an application



- **Decouples the application logic from the data layer**
- **Minimizes duplicate data access logic**

What is Docker?

Docker provides the ability to package and run an application in a loosely isolated environment called a container



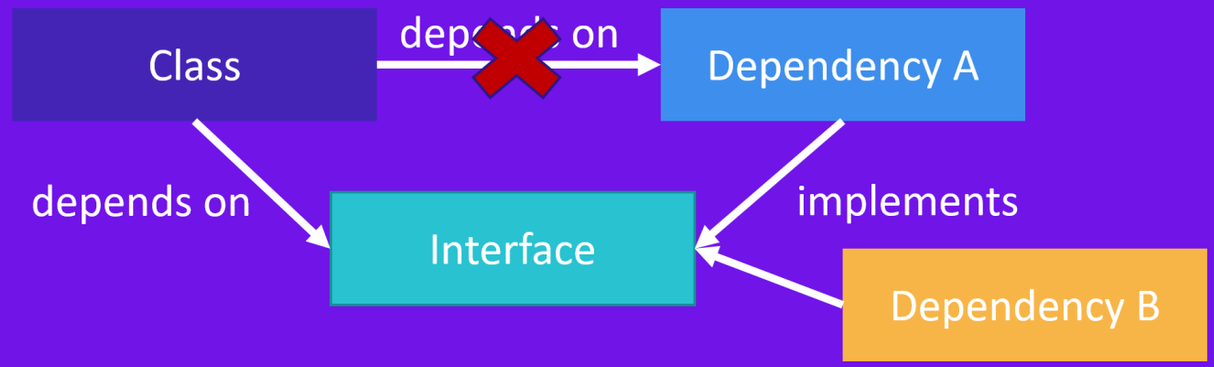
What is Dependency Injection?

Inject the repository dependency

```
public ItemsController(IItemsRepository itemsRepository)
{
    this.itemsRepository = itemsRepository;
}
```

```
ItemsRepository itemsRepository = new();
```

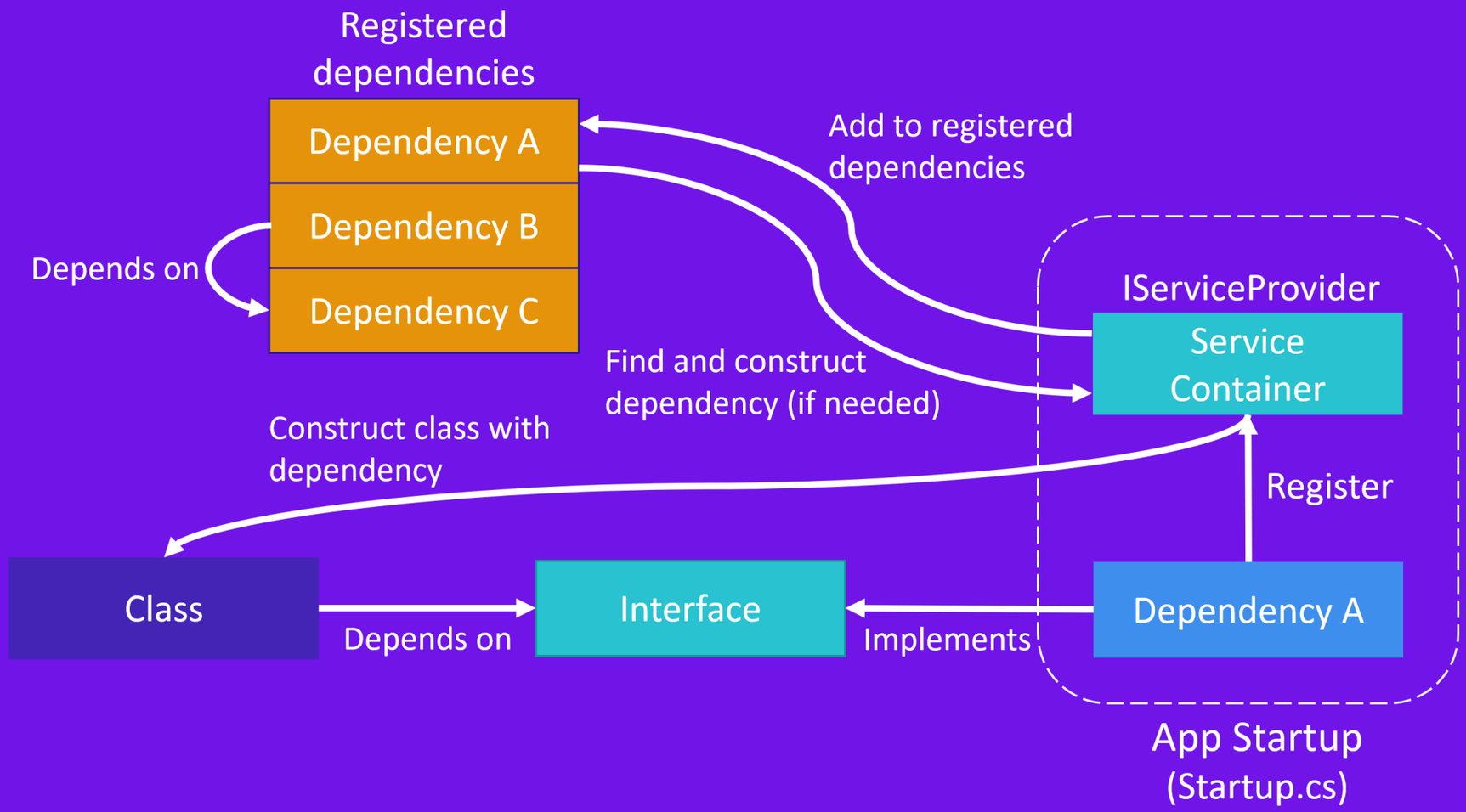
Dependency Inversion Principle



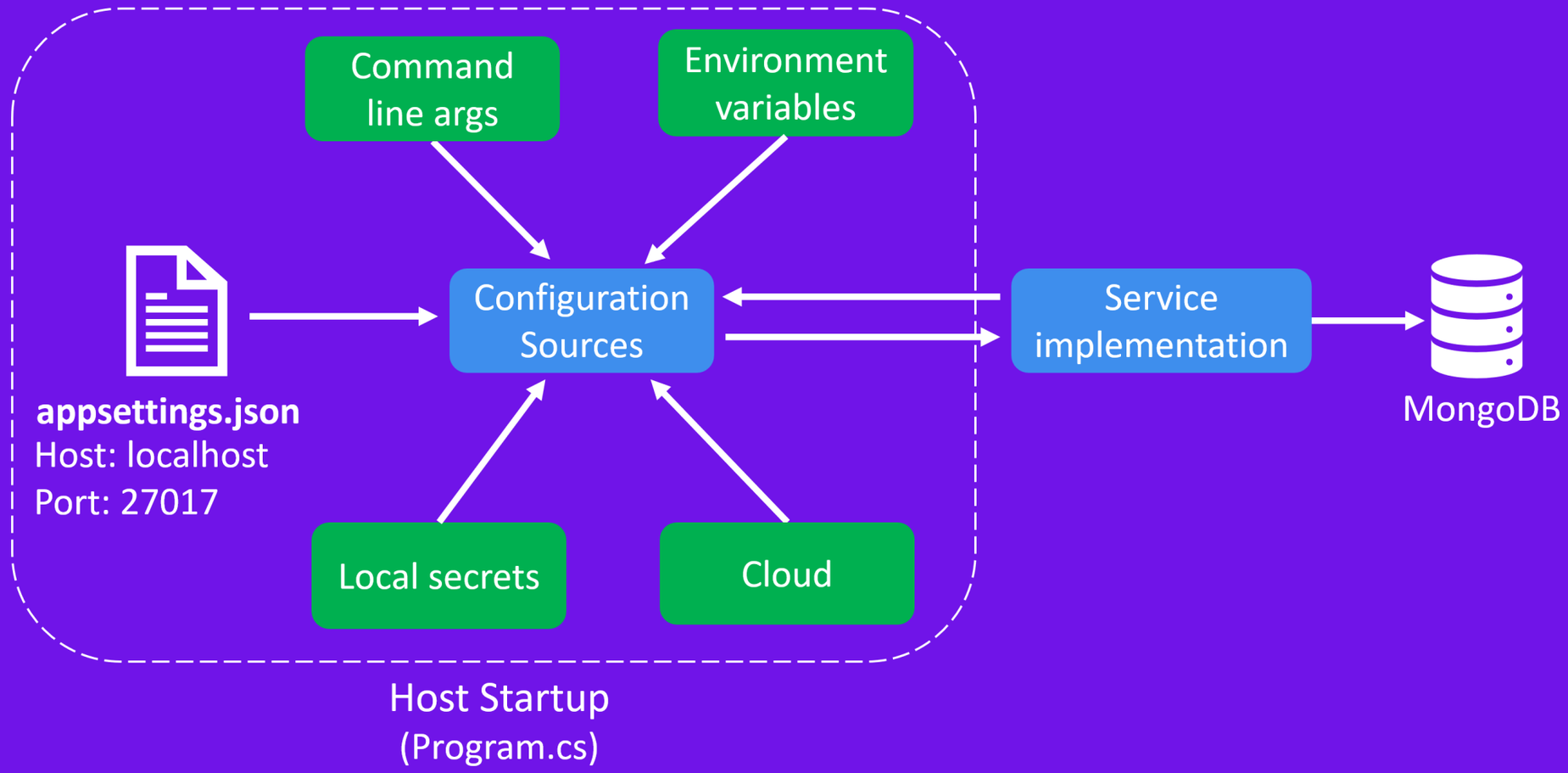
Benefits

- By having our code depend upon abstractions we are decoupling implementations from each other
- Code is cleaner, easier to modify and easier to reuse

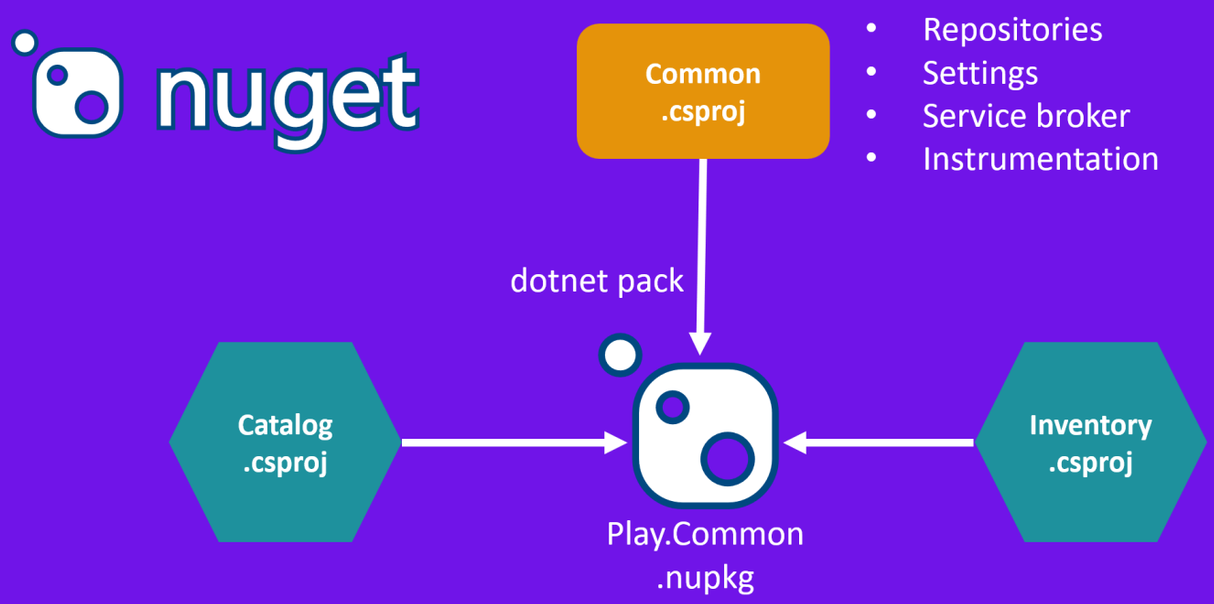
How to construct the dependencies?



ASP.NET Core Configuration



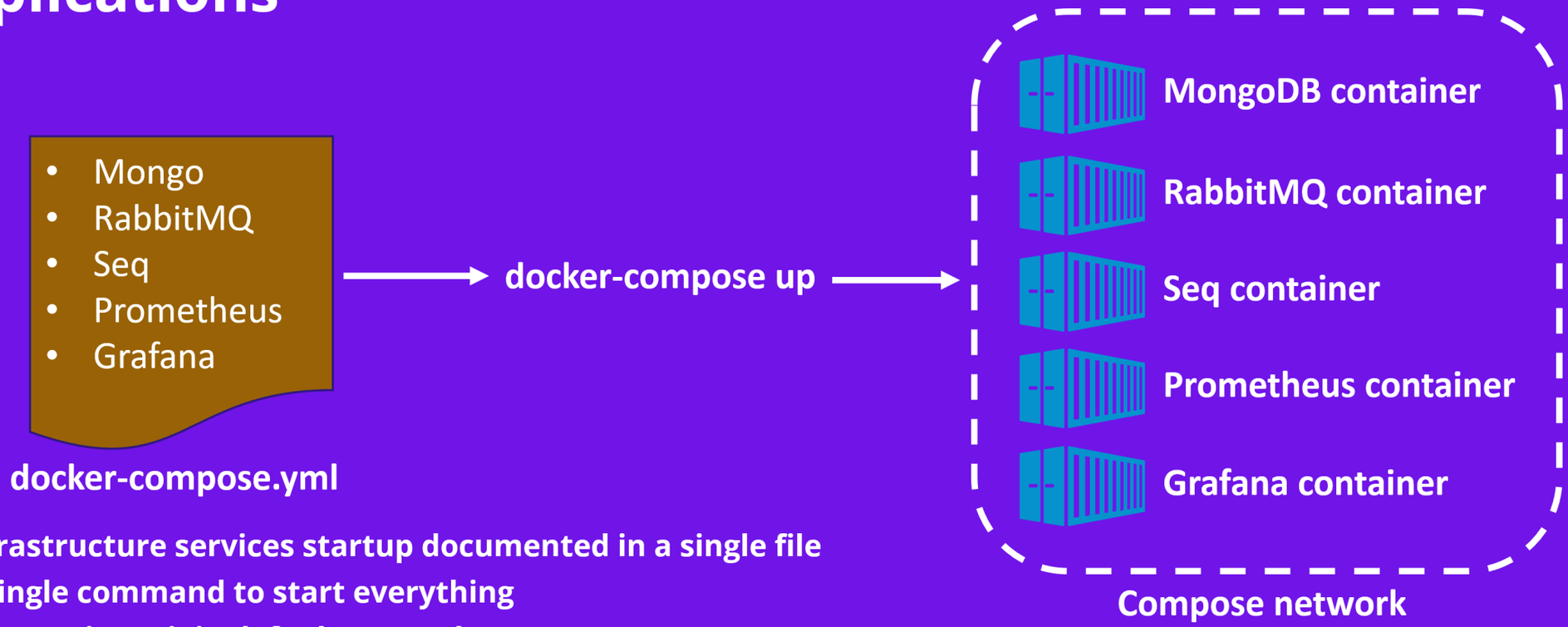
Reusing common code



- Don’t Repeat Yourself (DRY)
- Microservices should be independent of each other
- Each microservice should live in its own source control repository
- NuGet is the package manager for .NET
- A NuGet package is a single ZIP file (.nupkg) that contains files to share with others
- Microservice projects don’t need to know where NuGet packages are hosted
- The common code is now maintained in a single place
- The time to build new microservices is significantly reduced

What is Docker Compose?

A tool for defining and running multi-container Docker applications



- Infrastructure services startup documented in a single file
- A single command to start everything
- All containers join default network

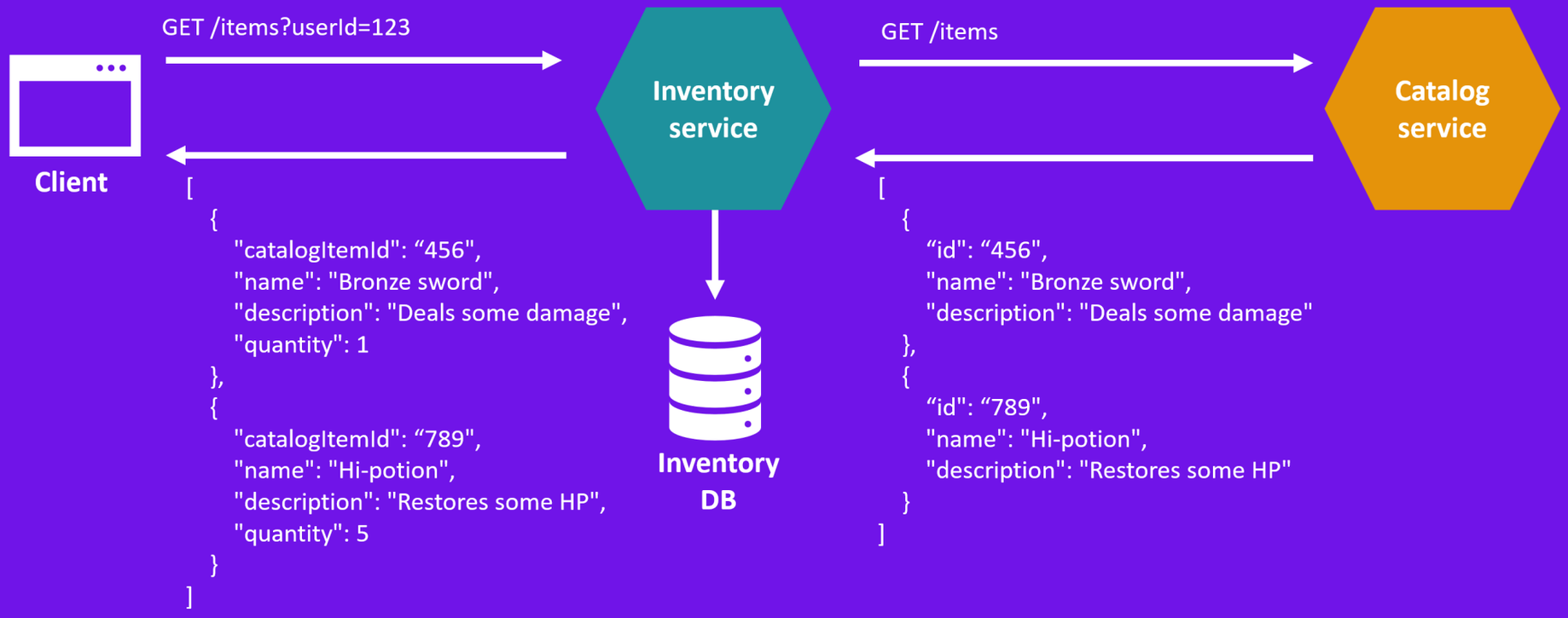
Microservice communication styles

Synchronous	Asynchronous
The client sends a request and waits for a response from the service	The client sends a request to the service but the response, if any, is not sent immediately

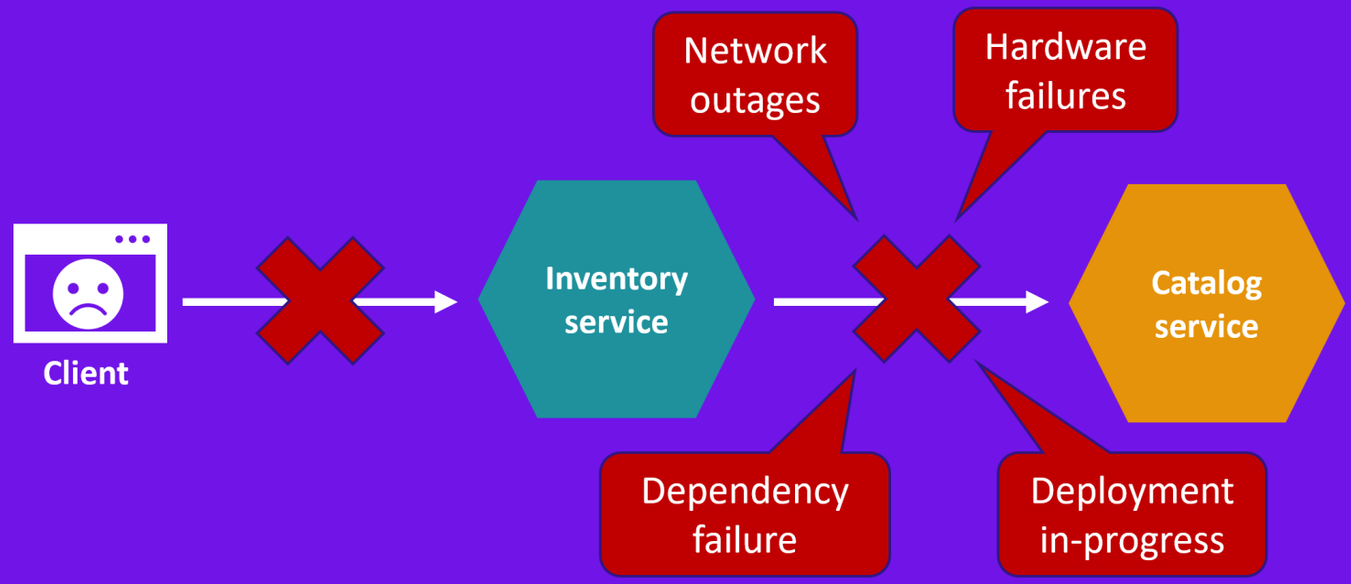
Synchronous communication style

- The client sends a request and waits for a response from the service
- The client cannot proceed without the response
- The client thread may use a blocking or non-blocking implementation (callback)
- REST + HTTP protocol is the traditional approach
- gRPC is an increasingly popular approach for internal inter-service communication

Implementing synchronous communication via REST + HTTP



Partial failures will happen



- In a distributed system, whenever a service makes a synchronous request to another service, there is an ever-present risk of partial failure
- You must design your service to be resilient to those partial failures

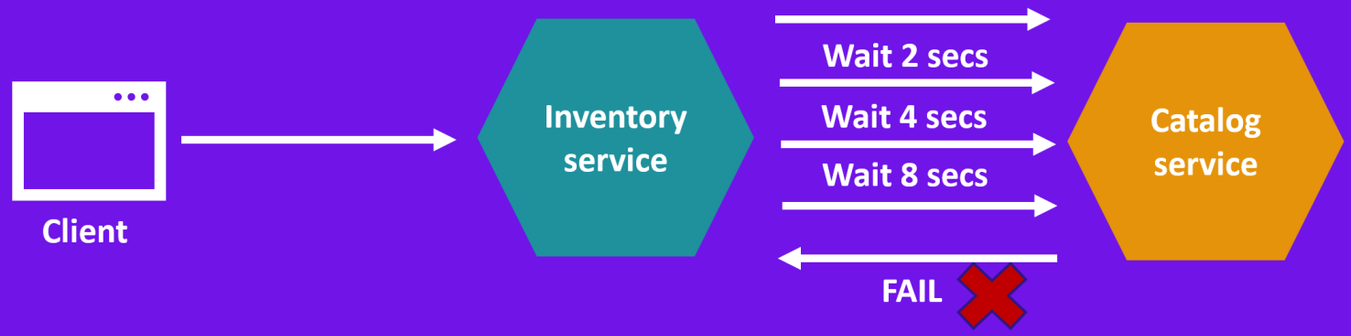
Setting timeouts

A service client should be designed not to block indefinitely and use timeouts

- **Use timeouts for a more responsive experience and to ensure resources are never tied up indefinitely**

Retries with exponential backoff

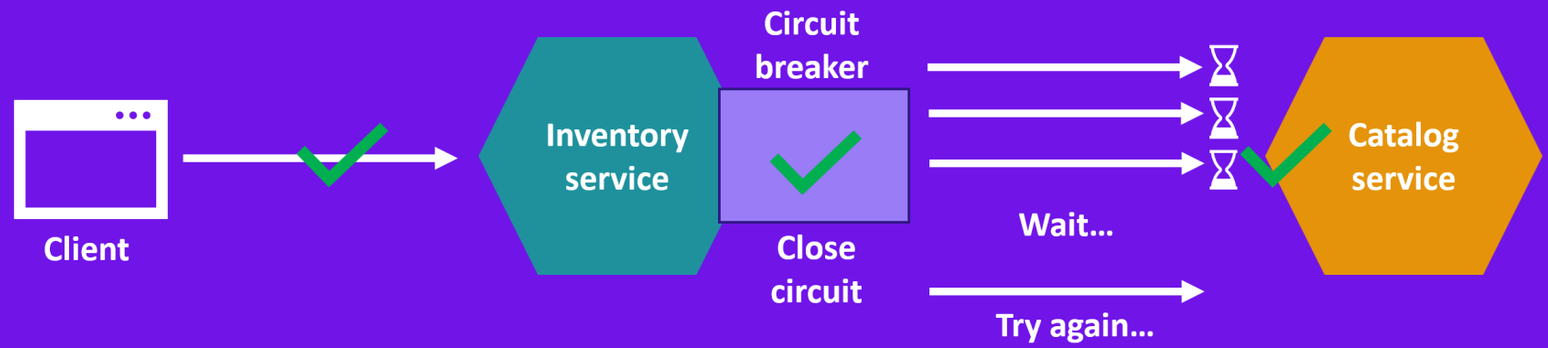
Performs call retries a certain number of times with a longer wait between each retry



- The failing dependency has an increasing amount of time to recover
- Avoids overwhelming the dependency

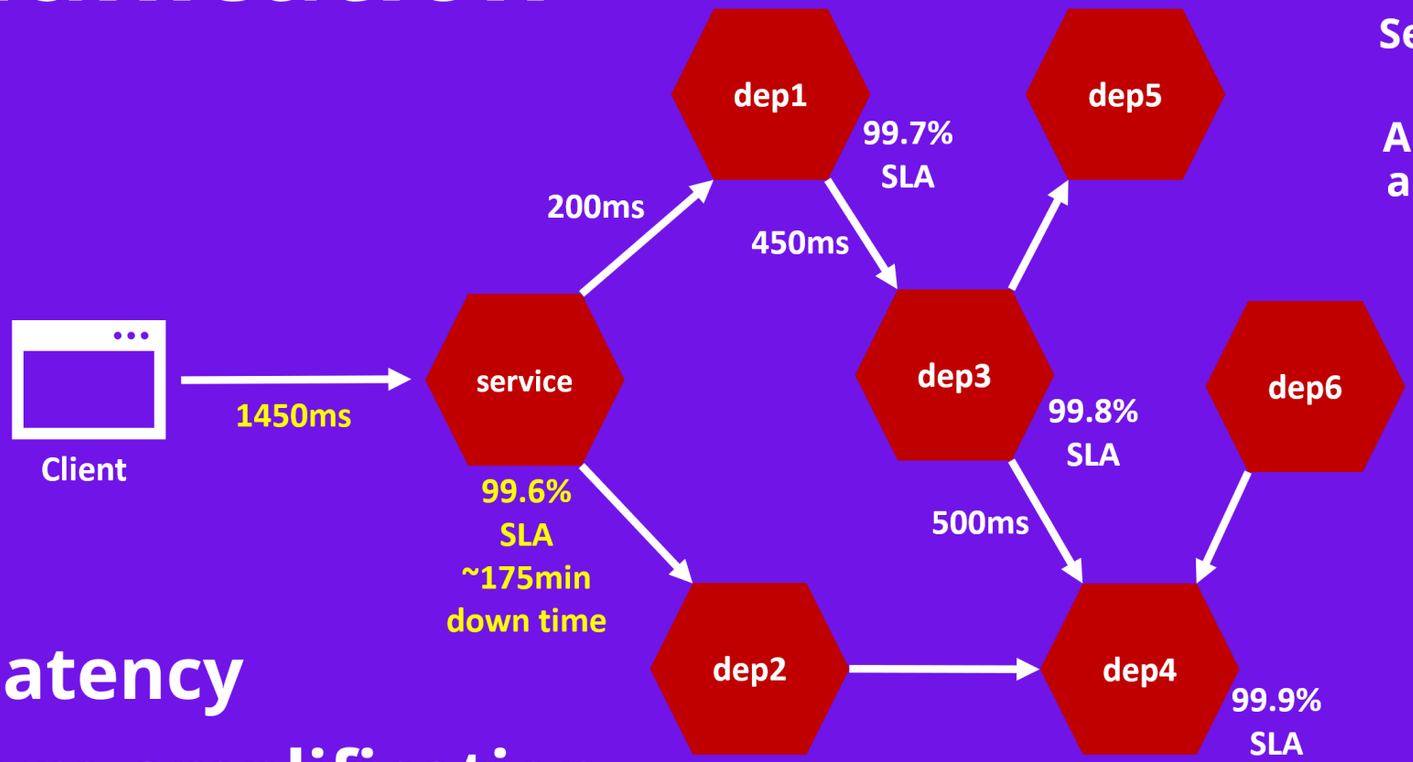
The circuit breaker pattern

Prevents the service from performing an operation that's likely to fail



- Prevents our service from reaching resource exhaustion
- Avoids overwhelming the dependency

The problem with synchronous communication



Service Level Agreement (SLA)

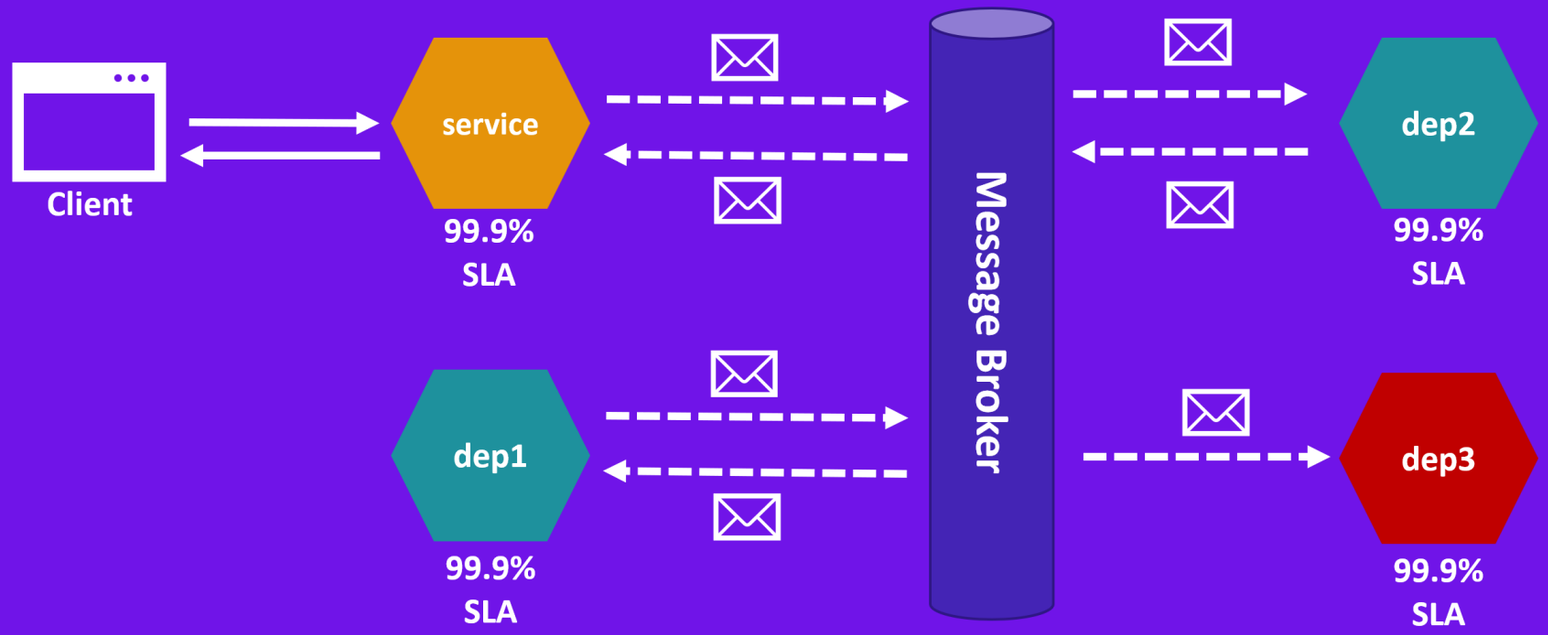
A commitment between a service provider and a client

- Increased latency
- Partial failure amplification
- Reduced SLA

Asynchronous communication style

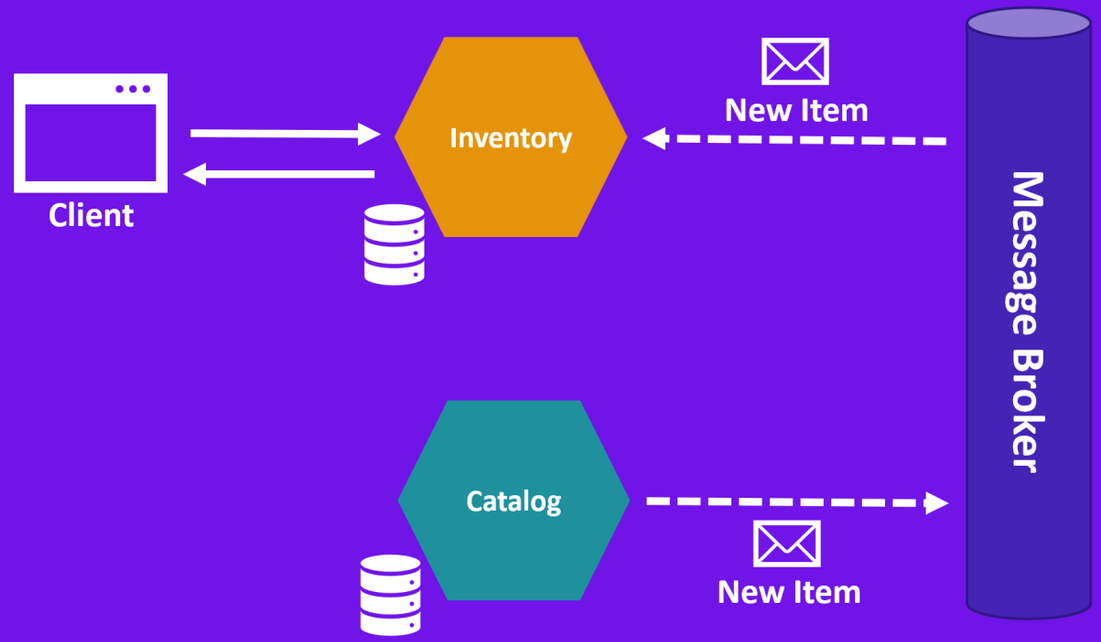
- The client does not wait for a response in a timely manner
- There might be no response at all
- Usually involves the use of a lightweight message broker
- Message broker has high availability
- Messages are sent to the message broker and could be received by:
  - A single receiver (asynchronous commands)
  - Multiple receivers (publish/subscribe events)

Microservices autonomy



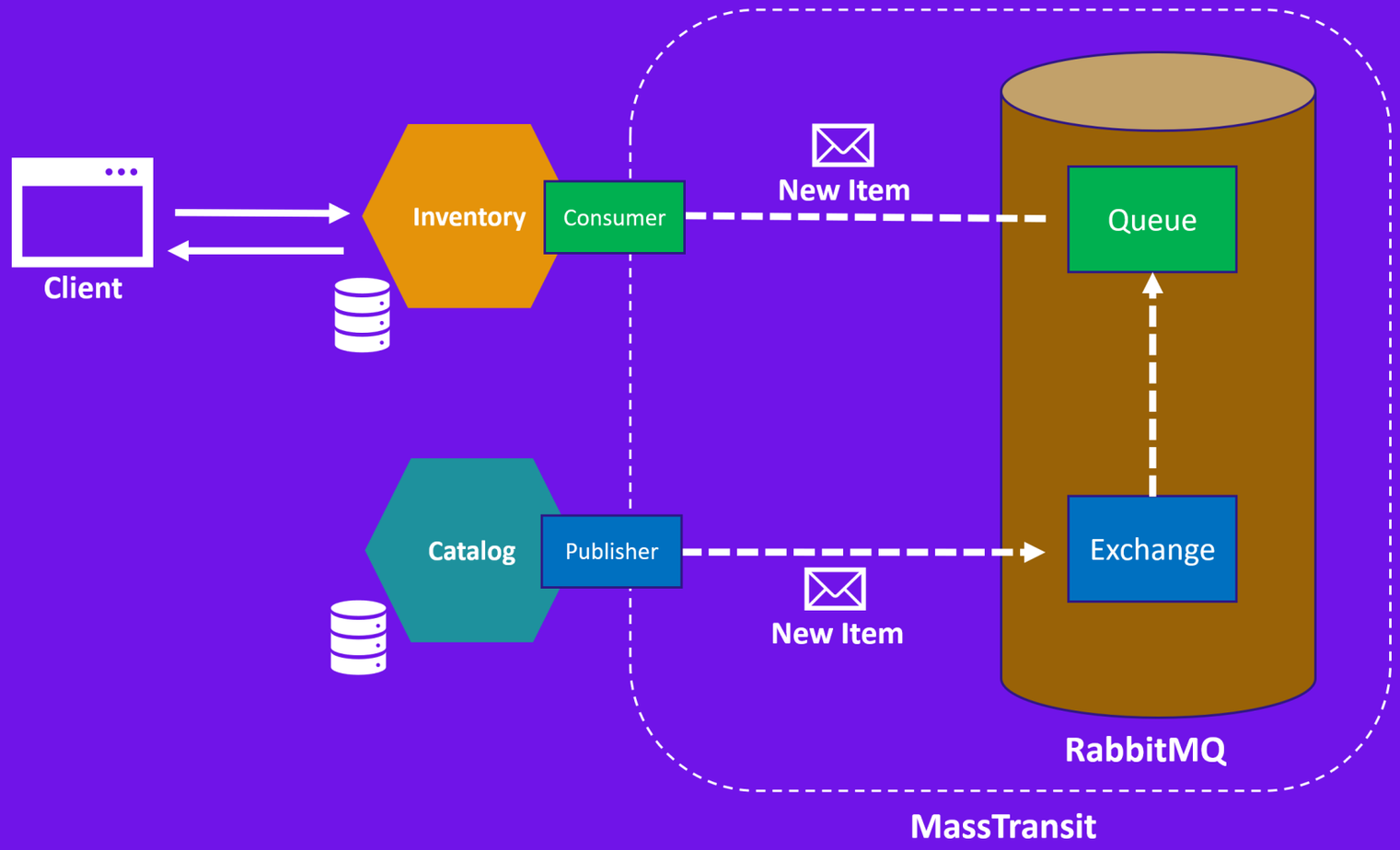
- Partial failures are not propagated
- Independent service SLA
- Microservice autonomy enforced

Asynchronous propagation of data



- Data is eventually consistent
- Preserves microservice autonomy
- Removes inter-service latency

Implementing asynchronous communication



RabbitMQ

Lightweight message broker that supports the AMQP protocol

MassTransit

Distributed app framework for .NET

Cross-Origin Requests

Same-origin policy

A web application can only request resources from the same origin the application was loaded from

Cross-Origin Resource Sharing (CORS)

CORS

Allows a server to indicate any other origins than its own from which a browser should permit loading of resources.

CORS Preflighted Requests

Browser

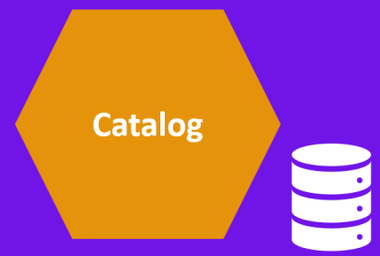
http://localhost:3000

OPTIONS https://localhost:5001/items
Headers:
Origin: http://localhost:3000
Access-Control-Request-Headers: content-type
Access-Control-Request-Method: POST

Headers:
Access-Control-Allow-Origin: http://localhost:3000
Access-Control-Allow-Headers: content-type
Access-Control-Allow-Methods: POST

POST https://localhost:5001/items
{"name":“Antidote","description":“Cures poison","price":9}
Headers:
Origin: http://localhost:3000

201 Created
Headers:
Access-Control-Allow-Origin: http://localhost:3000



https://localhost:5001

The need for a membership system

My Inventory

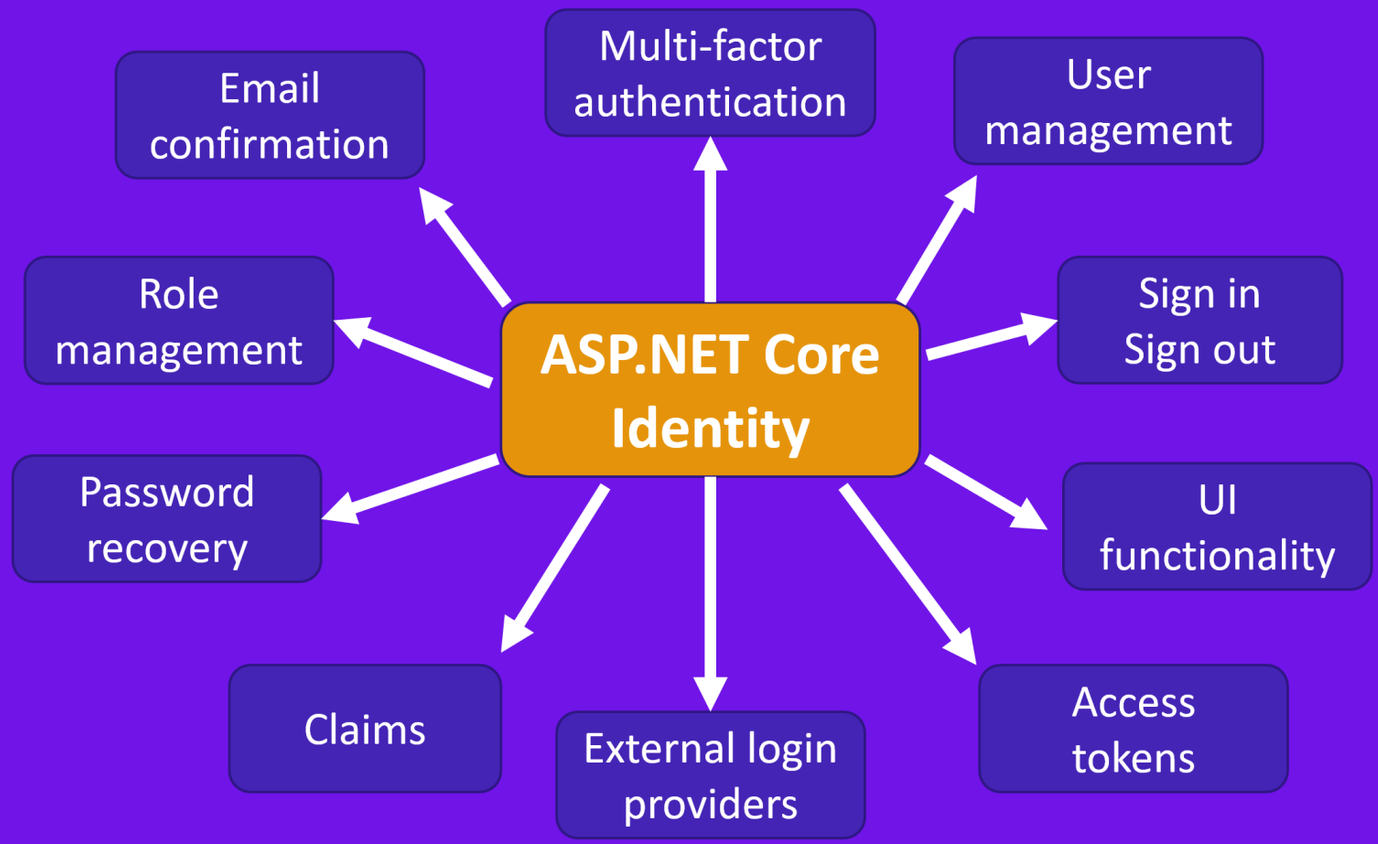
How much gil do I have?

How can I access the system?

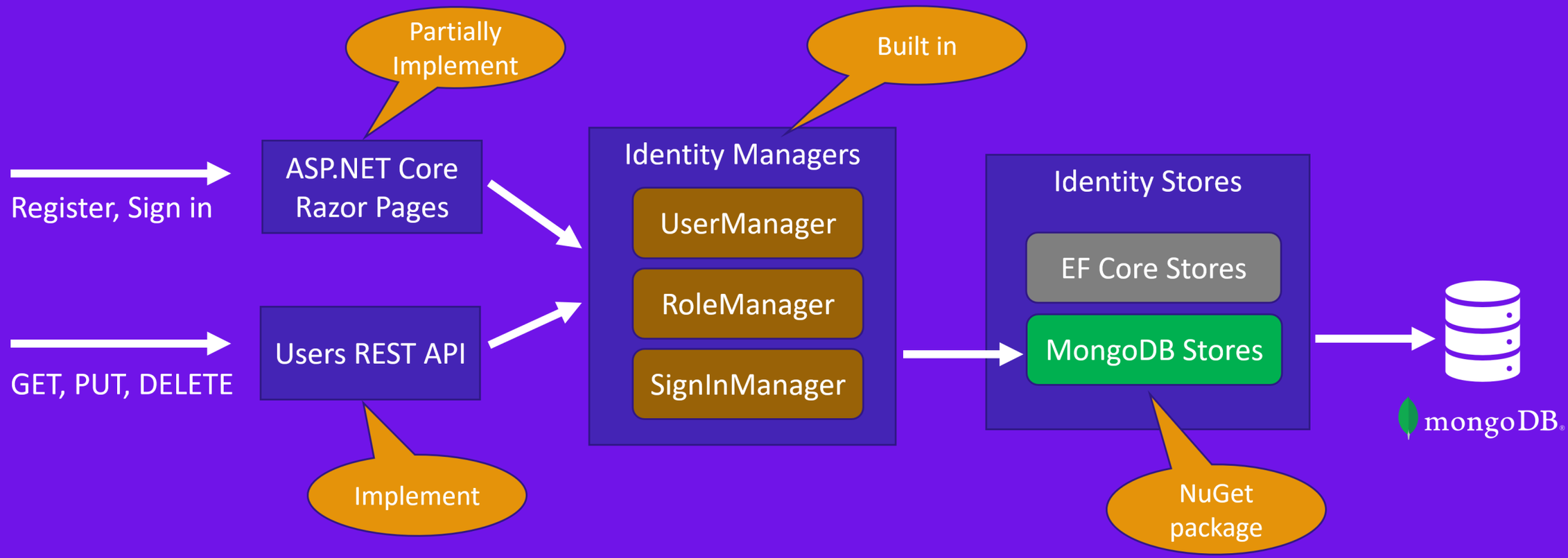
Is my data protected?

What can I do in the system?

ASP.NET Core Identity

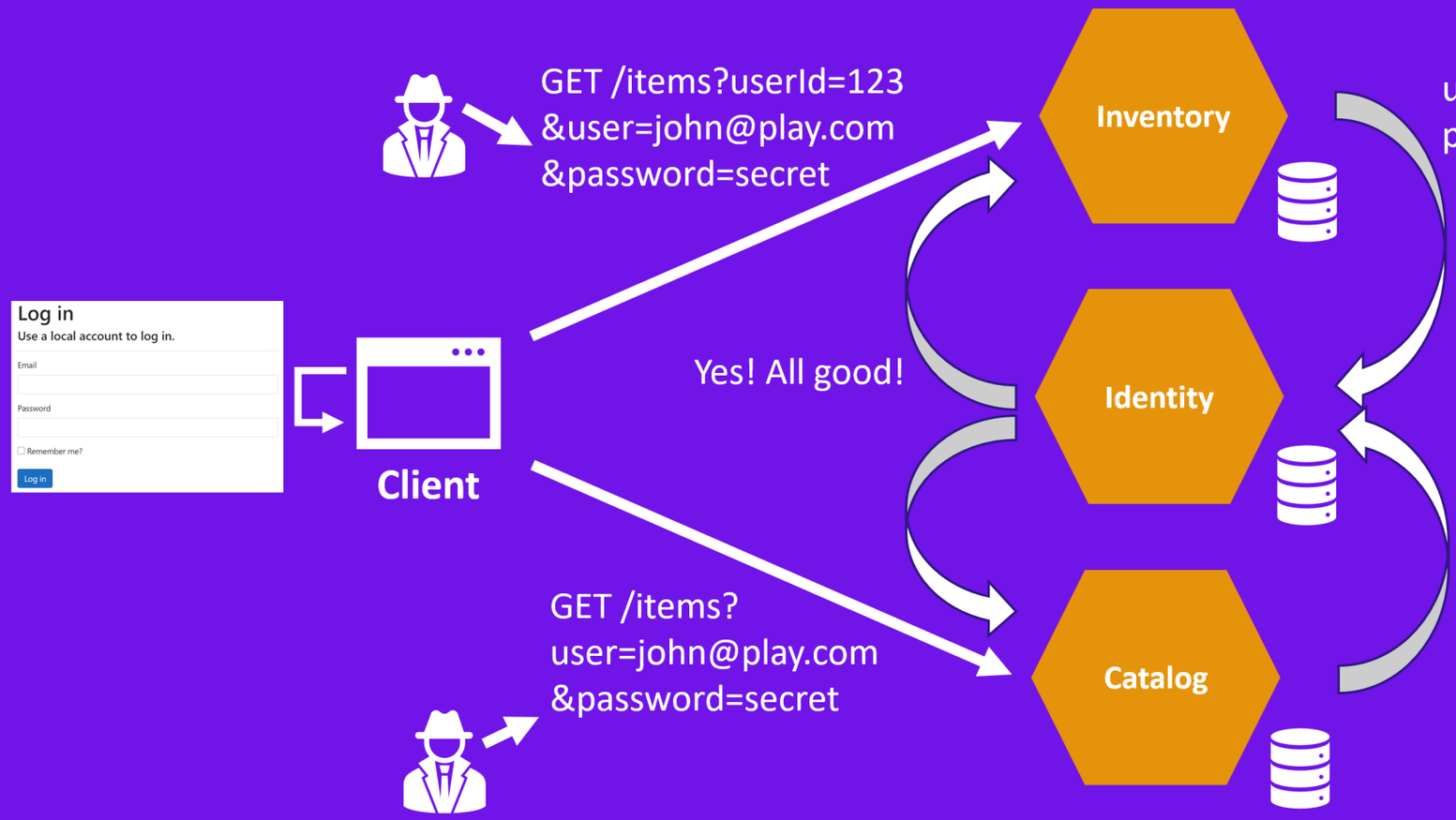


Implementing the Identity microservice



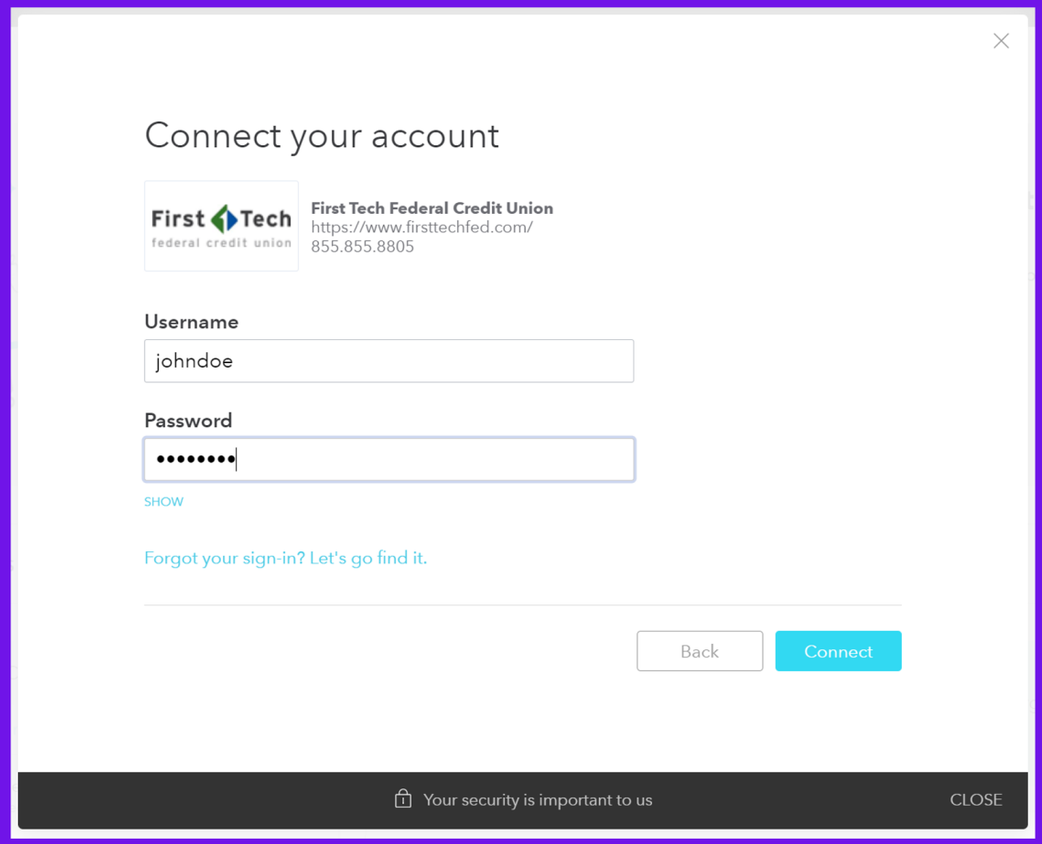
The need for authentication

Adding authentication



- An attacker with a package sniffer could grab the credentials
- Tools installed on the user's device could decrypt the URLs
- URLs would get logged on each microservice
- The size of the requests is bigger than needed
- The client asks for credentials every time or caches them
- Would overwhelm the Identity service and slow down things

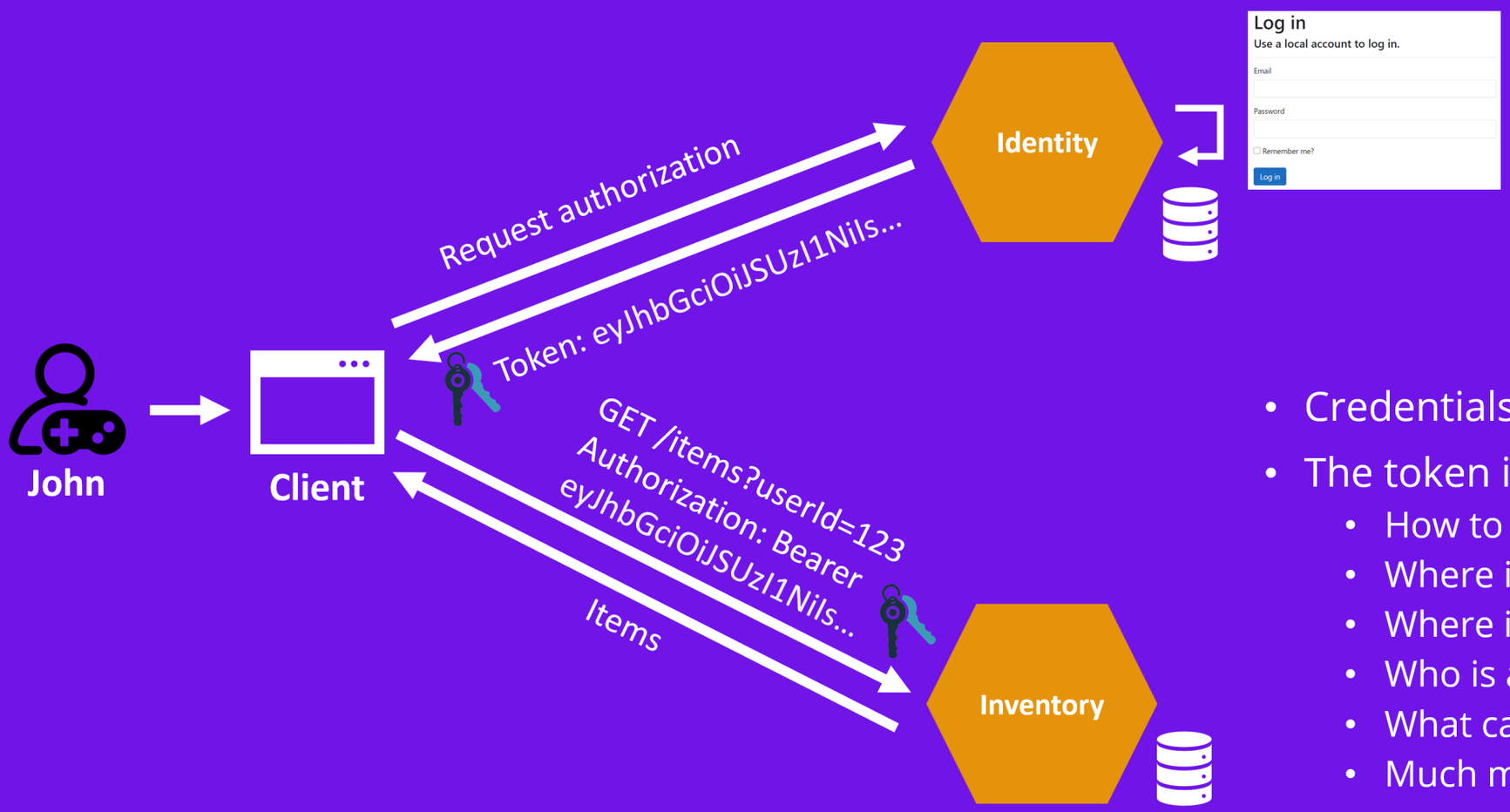
Giving away your credentials



*"Your login **user name** and **passwords** are **stored** securely in a **separate database** using multi-layered hardware and software encryption. We only store the information needed to save you the trouble of updating, syncing or uploading financial information manually."*

- Bad idea!
- Only your bank should have your credentials, no one else
- What exactly am I allowing it to do with my bank account?

Enter token-based security

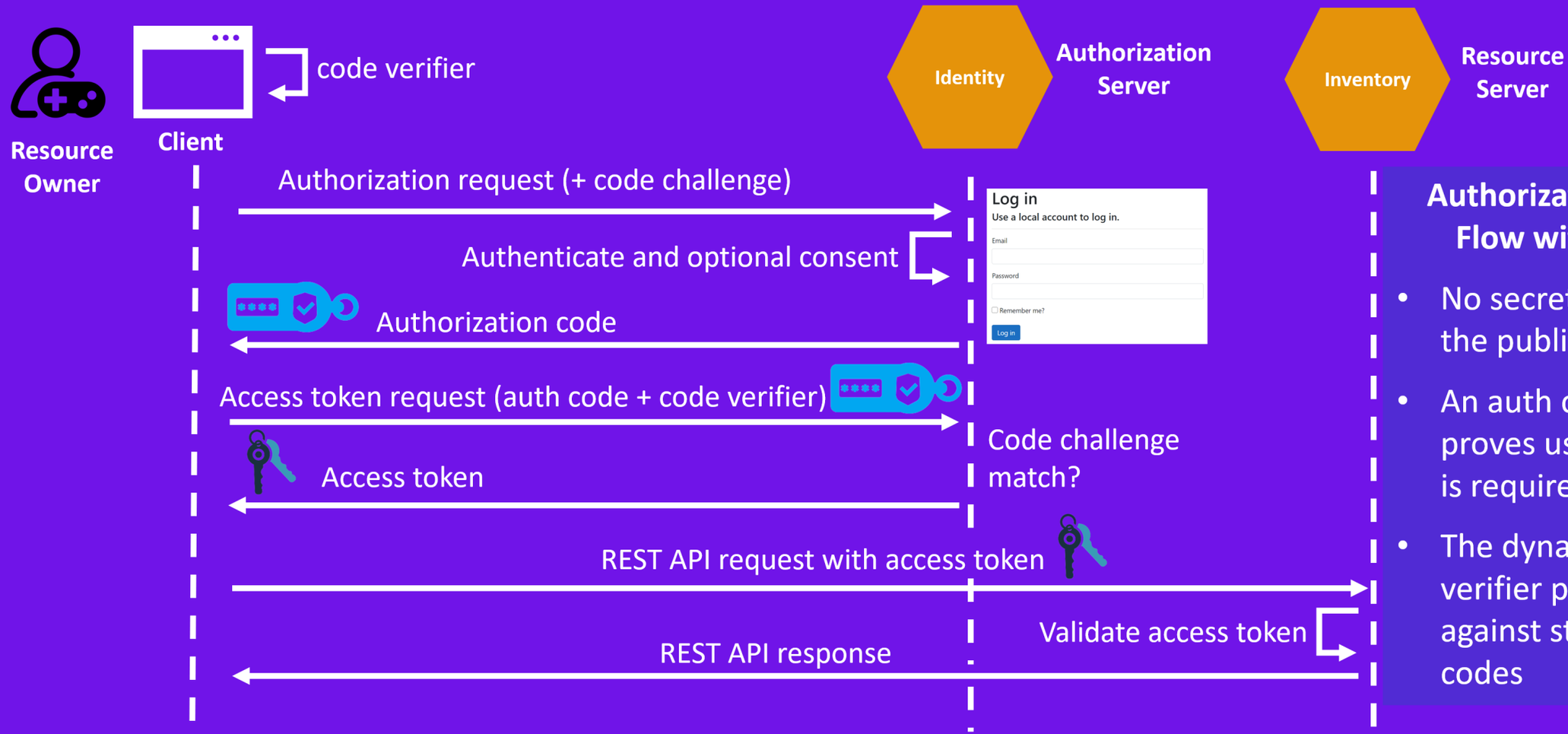


- Credentials are sent only once
- The token includes info about:
  - How to verify it
  - Where it came from
  - Where it can be used
  - Who is authorized
  - What can be done with it
  - Much more

Introducing OAuth 2.0

OAuth 2.0 is the industry-standard protocol for authorization, allowing a website or application to access resources hosted by other web apps on behalf of a user

The OAuth 2.0 authorization flow



Authorization Code Flow with PKCE

- No secret stored in the public client
- An auth code that proves user consent is required
- The dynamic code verifier protects against stolen auth codes

The Authorization Request

Authorization server

Authorization endpoint

GET **https://localhost:5003/connect/authorize**?

response_type=code — "Auth server, I need an authorization code"

&**client_id**=gameclient — "I am the game client"

&**scope**=Inventory — "I need access to the Inventory API"

"I registered with you using this url"

&**redirect_uri**=https://localhost:3000/authentication/login-callback

&**code_challenge**=Y2RfVUD4AIYO-XEW5lAejoRtCw3DfyCJZsSaLpgxDX4

&**code_challenge_method**=S256 — "The hash method of my code challenge"

"I generated this out of my code verifier"

The Access Token Request

Authorization server

Token endpoint

POST **https://localhost:5003/connect/token**

grant_type=authorization_code

"Auth server, exchange my code for an access token"

&**client_id**=gameclient

"I am the game client"

"I registered with you using this url"

&**redirect_uri**=https://localhost:3000/authentication/login-callback

&**code**=5DBC1EC9E02391AB561F1727B4E546BD...

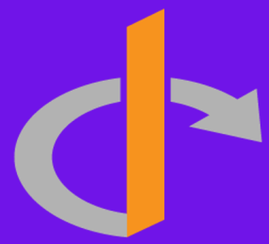
"Here's the authorization code you gave me"

&**code_verifier**=IDchfXpYPJH8yEFMyDgKSIXHpFWr...

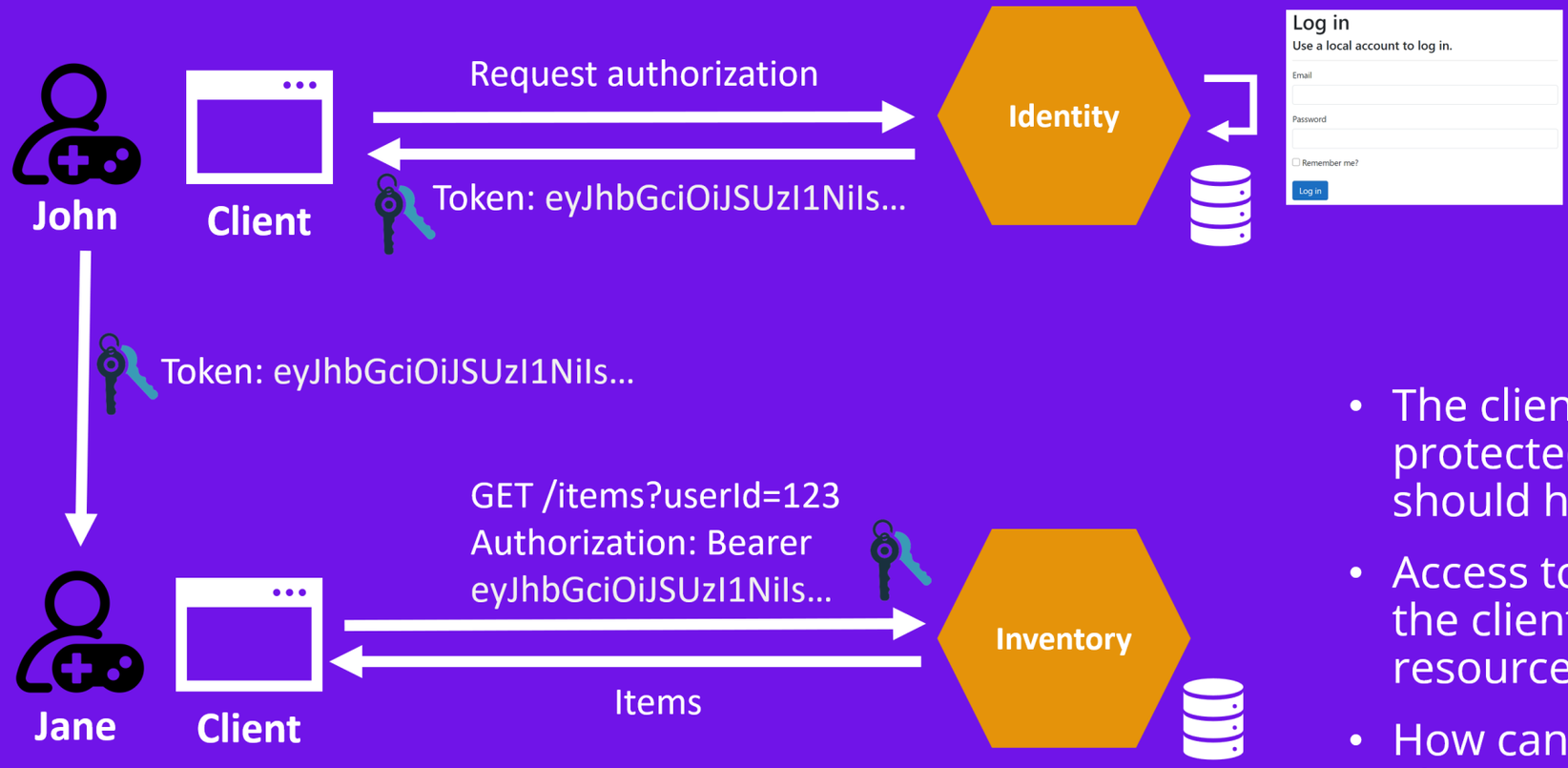
"My original code verifier that you should be able to confirm"

What is OpenID Connect?

OpenID Connect is a simple identity layer on top of the OAuth 2.0 protocol that allows clients to verify the identity of the End-User

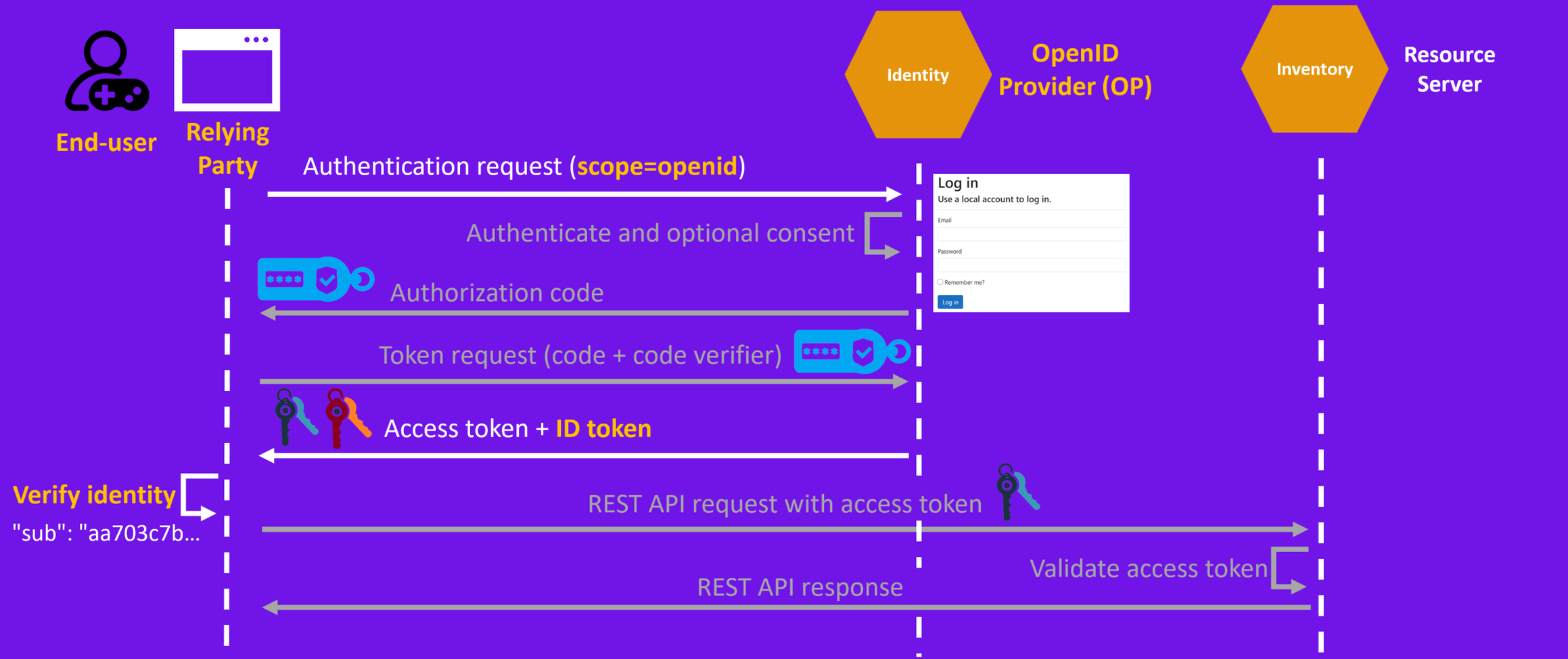


OAuth 2.0 is not enough

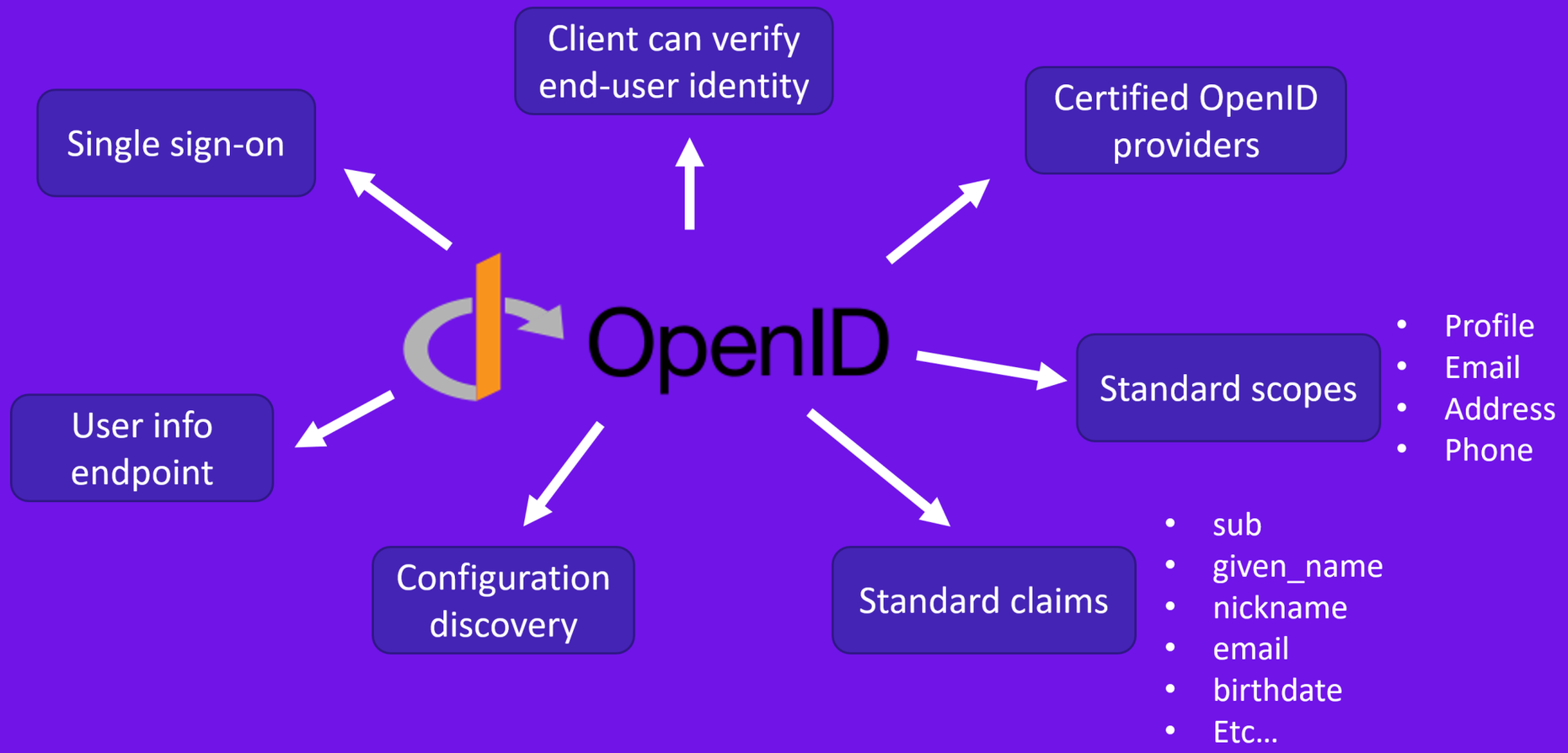


- The client app got access to protected resources when it should have not
- Access tokens are not meant for the client app but for protected resources
- How can the client app verify the identity of the end user?

The OpenID Connect authentication flow



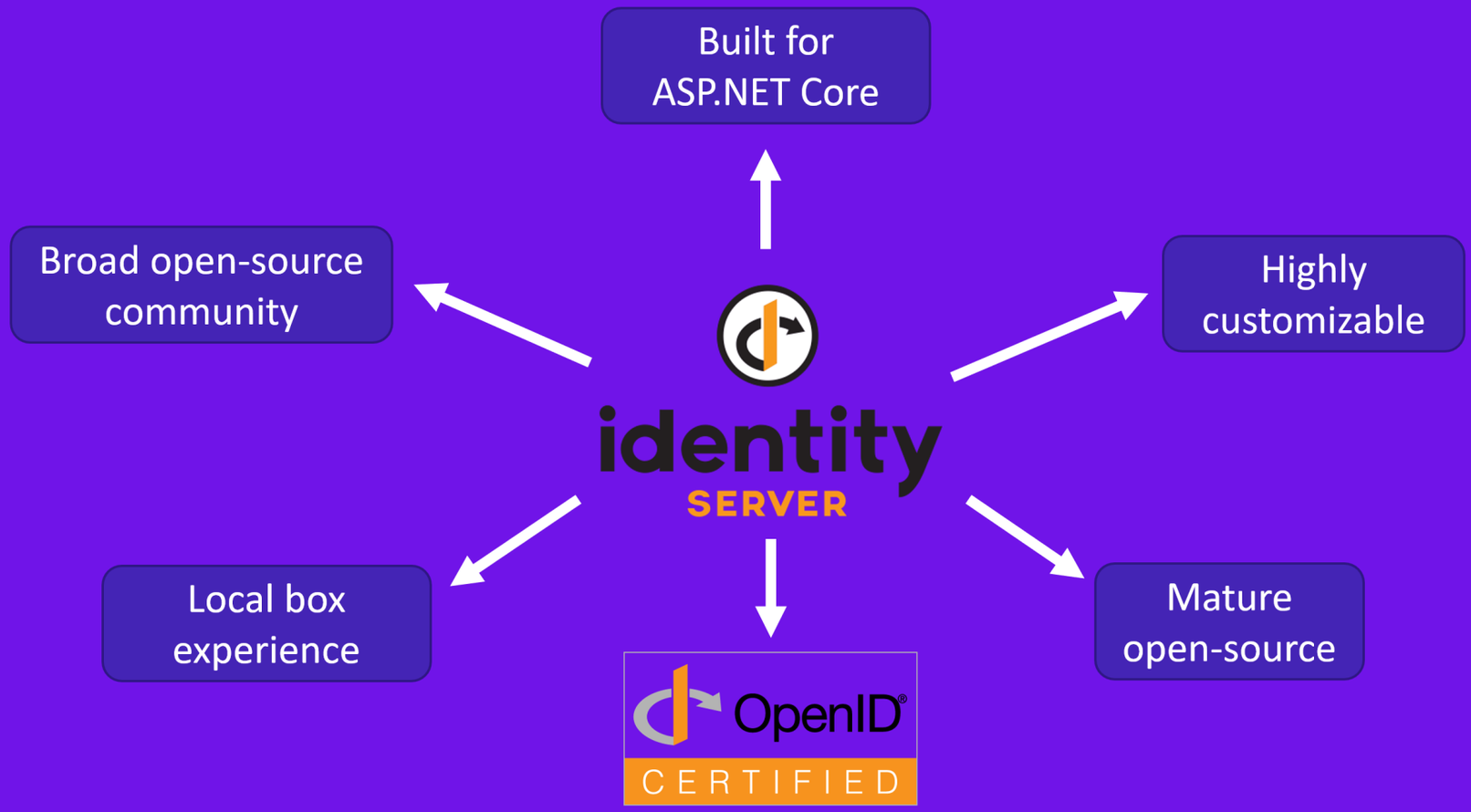
What OpenID Connect adds



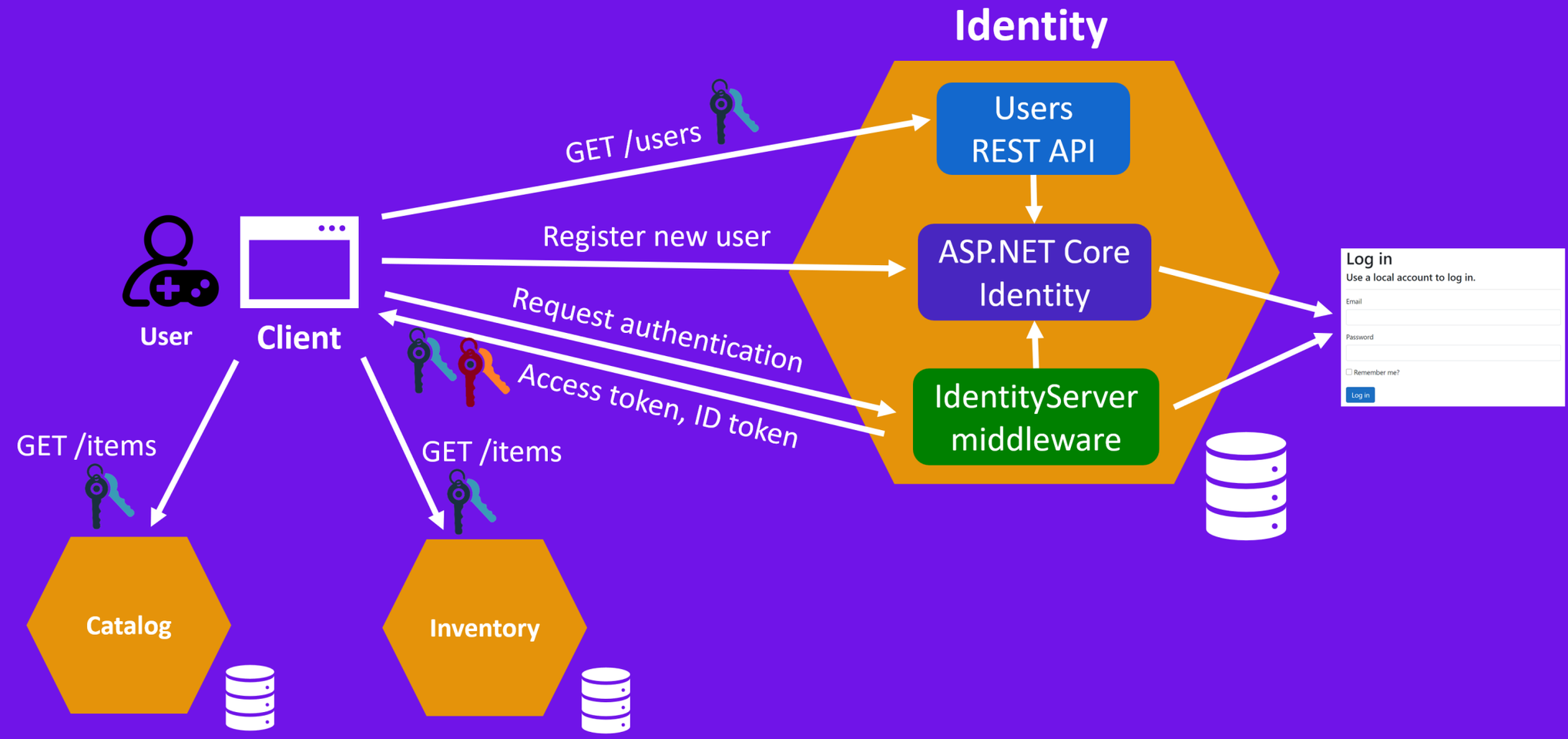
Introducing IdentityServer

An open-source, standards-compliant, and flexible OpenID Connect and OAuth 2.0 framework for ASP.NET Core

Why IdentityServer?



The Identity microservice revisited



What is a JSON Web Token (JWT)?

JSON Web Token (JWT) is a compact, URL-safe means of representing claims to be transferred between two parties.

A signed JWT is known as a JSON Web Signature (JWS)

JSON Web Token (JWT) structure

JWT Header

Field	Sample Value	Description
alg	RS256	Signing algorithm
typ	jwt at+jwt	Type of token jwt: JSON Web Token at+jwt: JWT access token
kid	39629176BDF969722B7D83B77500D83C	Identifier of the key used to sign the token

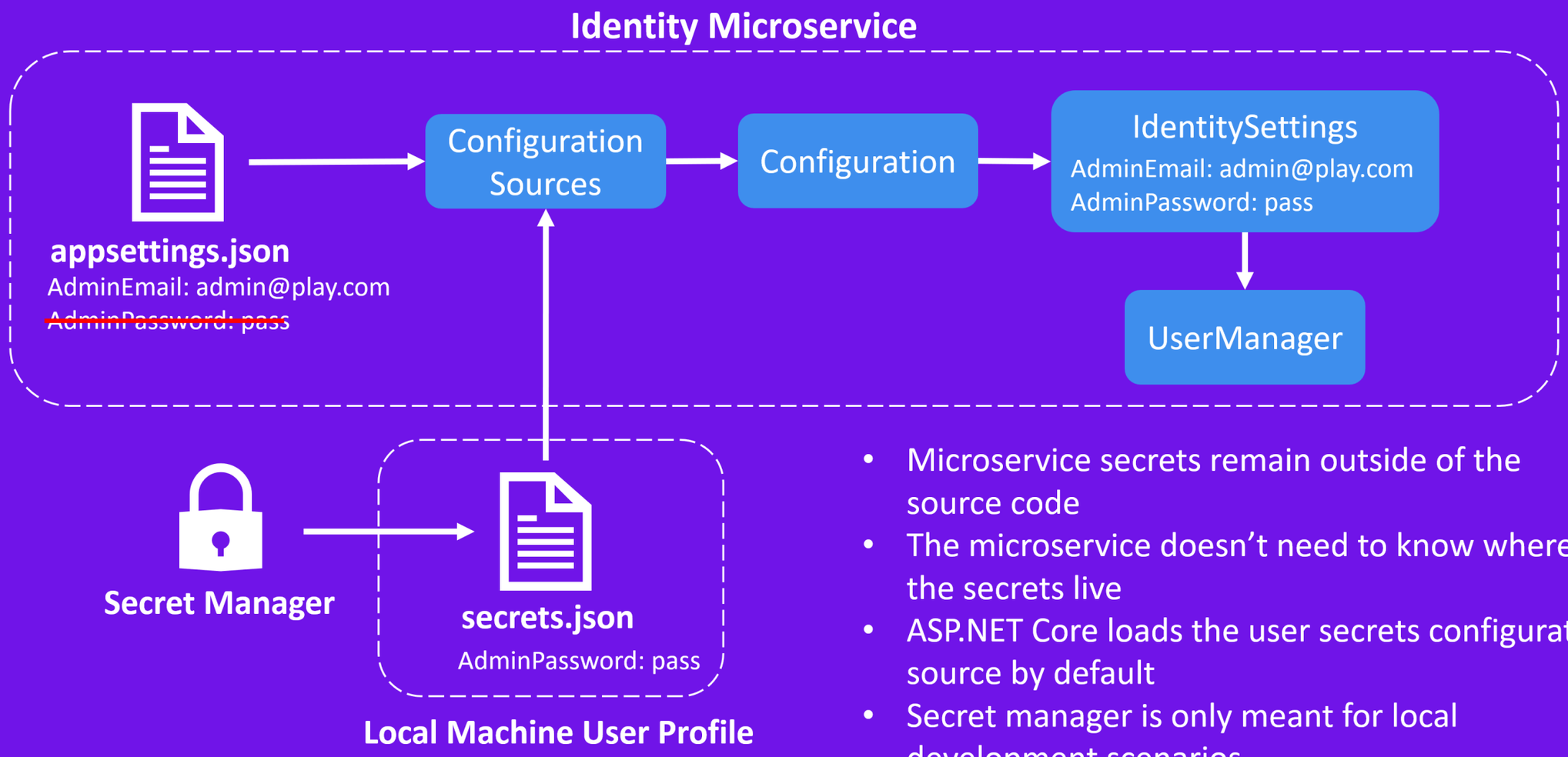
JWT Payload

Field	Sample Value	Description	Used in
iss	https://localhost:5003	Issuer: Who created and signed the token.	ID token Access token
aud	Catalog, Inventory, postman	Audience: Who the token is intended for.	ID token Access token
client_id	postman	The identifier of the OAuth client	Access token
sub	ccf2b778-2cc9-4478-82ca-2780a82f63ce	Subject. The principal that is the subject of the JWT. The Id of the signed in user in the Identity DB.	ID token Access token
scope	openid, profile, catalog.readaccess, inventory.fullaccess	The type of access granted to the client.	Access token
at_hash	dzJsGOPnXdR-haI8OtRSjA	Access Token hash value: Used to correlate the id token with the access_token	ID token
amr	pwd, mfa, pin	Authentication Methods References. Array of strings that identify the used authentication methods.	ID token Access token

JWT Payload (continued)

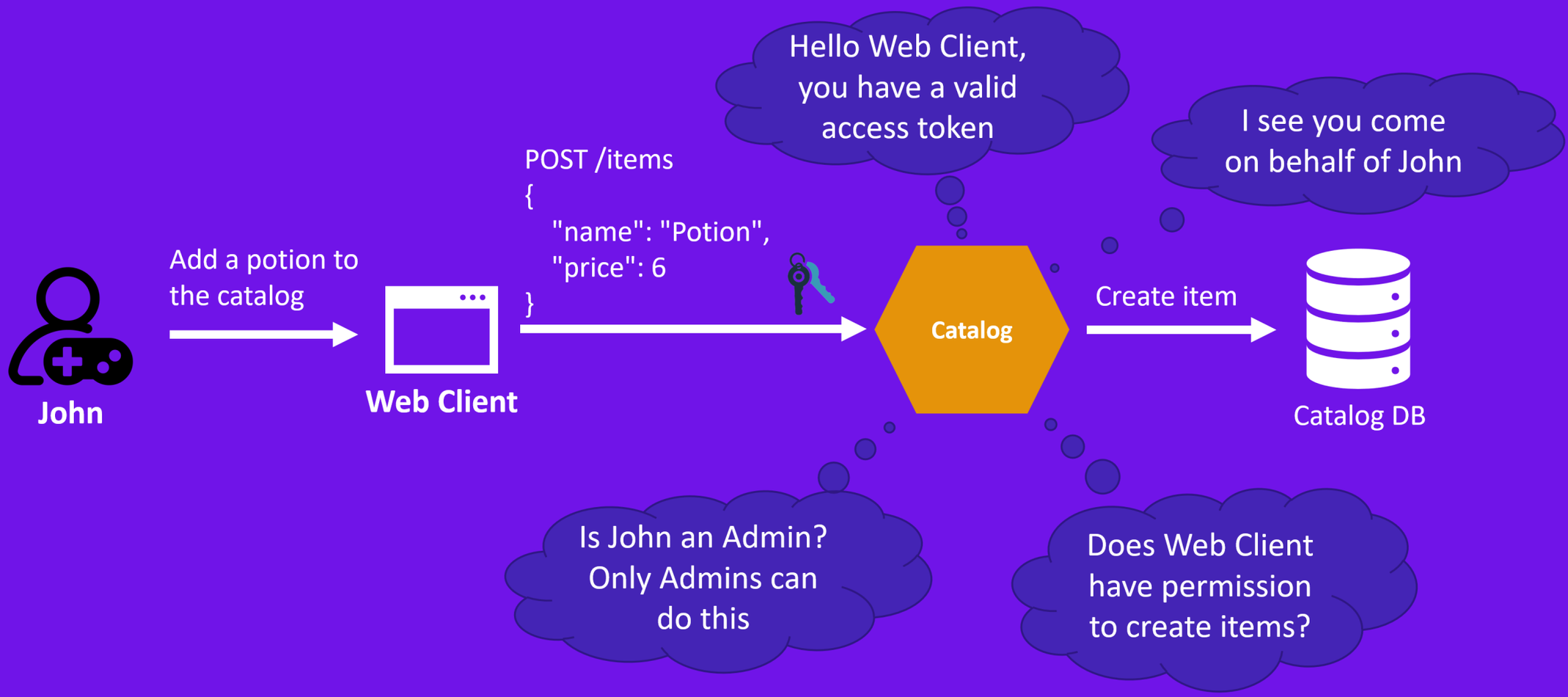
Field	Sample Value	Description	Used in
nbf	1615097311	Not valid before. The time before which the JWT MUST NOT be accepted for processing.	ID token Access token
exp	1615100911	Expiration time after which the JWT MUST NOT be accepted for processing	ID token Access token
iat	1617140901	The time at which the JWT was issued	ID token Access token
auth_time	1615093118	The time at which the end user last authenticated	ID token Access token
idp	local	Identity provider name	ID token Access token
sid	9927C19C7742C9323BEA6C1D01575995	Unique identifier of the session of the end user on a particular device/user agent	ID token Access token
jti	93CC732B92ECEA50E7DAEA8E7E85A7C3	The unique identifier of the JWT. Prevents the JWT from being replayed.	Access token

The .NET Secret Manager



- Microservice secrets remain outside of the source code
- The microservice doesn't need to know where the secrets live
- ASP.NET Core loads the user secrets configuration source by default
- Secret manager is only meant for local development scenarios

The need for authorization



ASP.NET Core role-based authorization

- Allows limiting access to resources based on the roles the user belongs to
- A role is a claim associated to a user

Player

- Get his own inventory bag
- Get his current amount of gil

Admin

- Get/update anybody's inventory bag
- Get/update the details of any user
- Get/update the items Catalog

ASP.NET Core claims-based authorization

- Allows limiting access to resources based on the claims presented in the access token
- Access is granted through policies that group a series of claims requirements

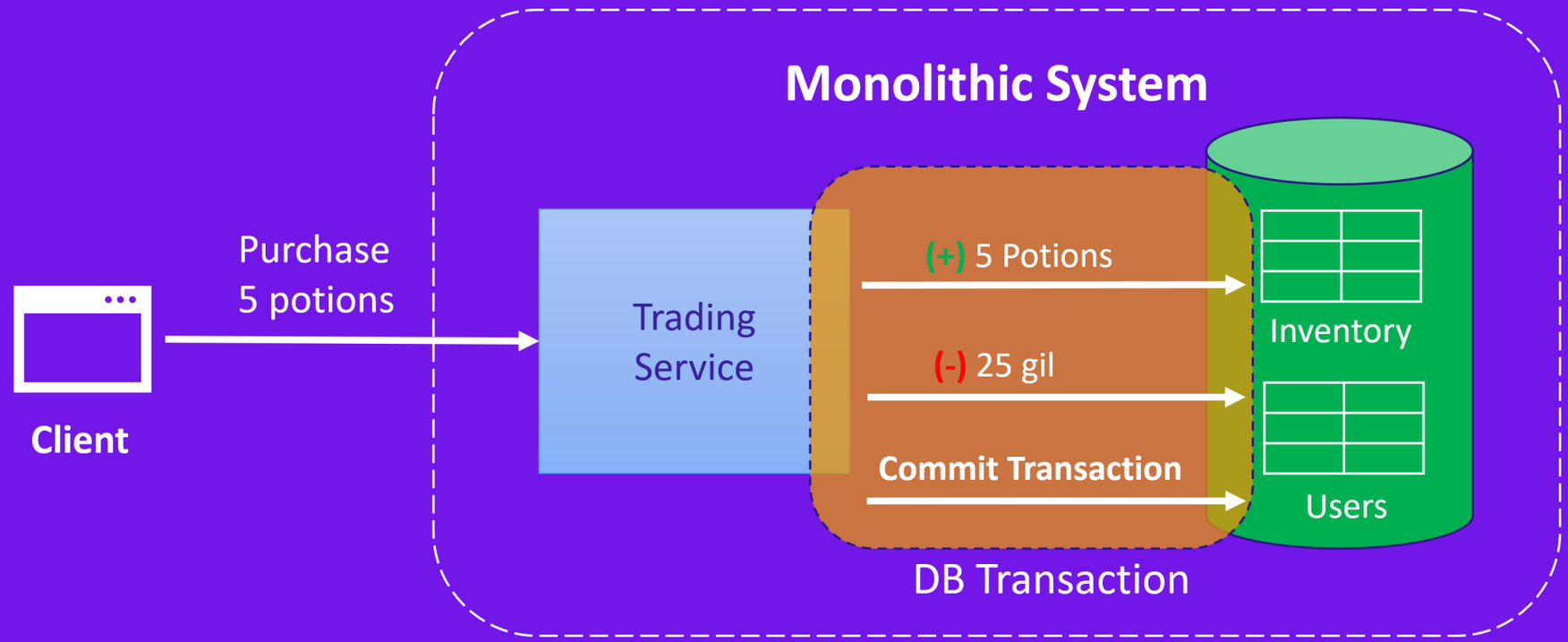
Policy	Requirements
Read	Role: Admin Scope: catalog.readaccess or catalog.fullaccess
Write	Role: Admin Scope: catalog.writeaccess or catalog.fullaccess

The need for transactions

Transaction

- A set of operations that should be treated as a single unit
- Either all operations complete successfully or all of them are rolled back

Transactions in a monolith



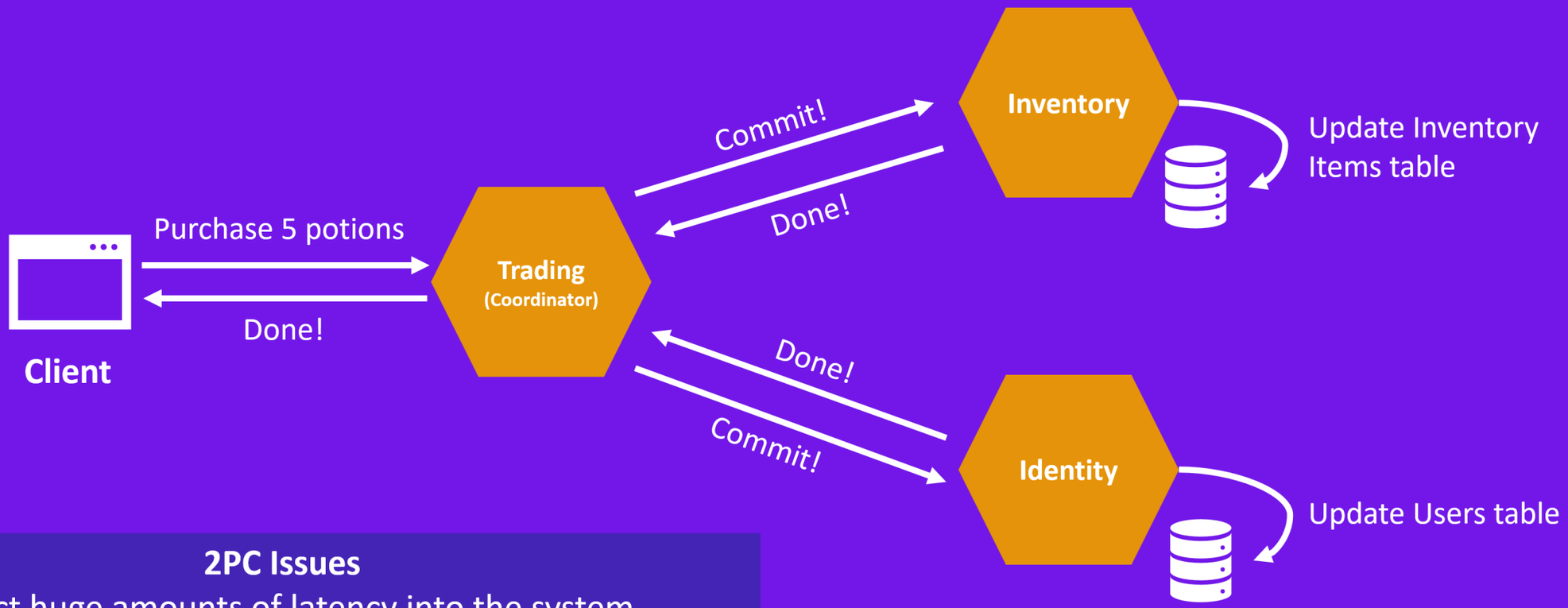
ACID Transaction	
Atomicity:	**All operations complete or all fail**
Consistency:	The database is left in a valid, consistent state
Isolation:	Operations execute at the same time without interfering
Durability:	Once the transaction completes, data won't get lost if the are system failures

What is a distributed transaction?

"A distributed transaction is a database transaction in which two or more network hosts are involved"

WIKIPEDIA
The Free Encyclopedia

Two-Phase Commits



2PC Issues

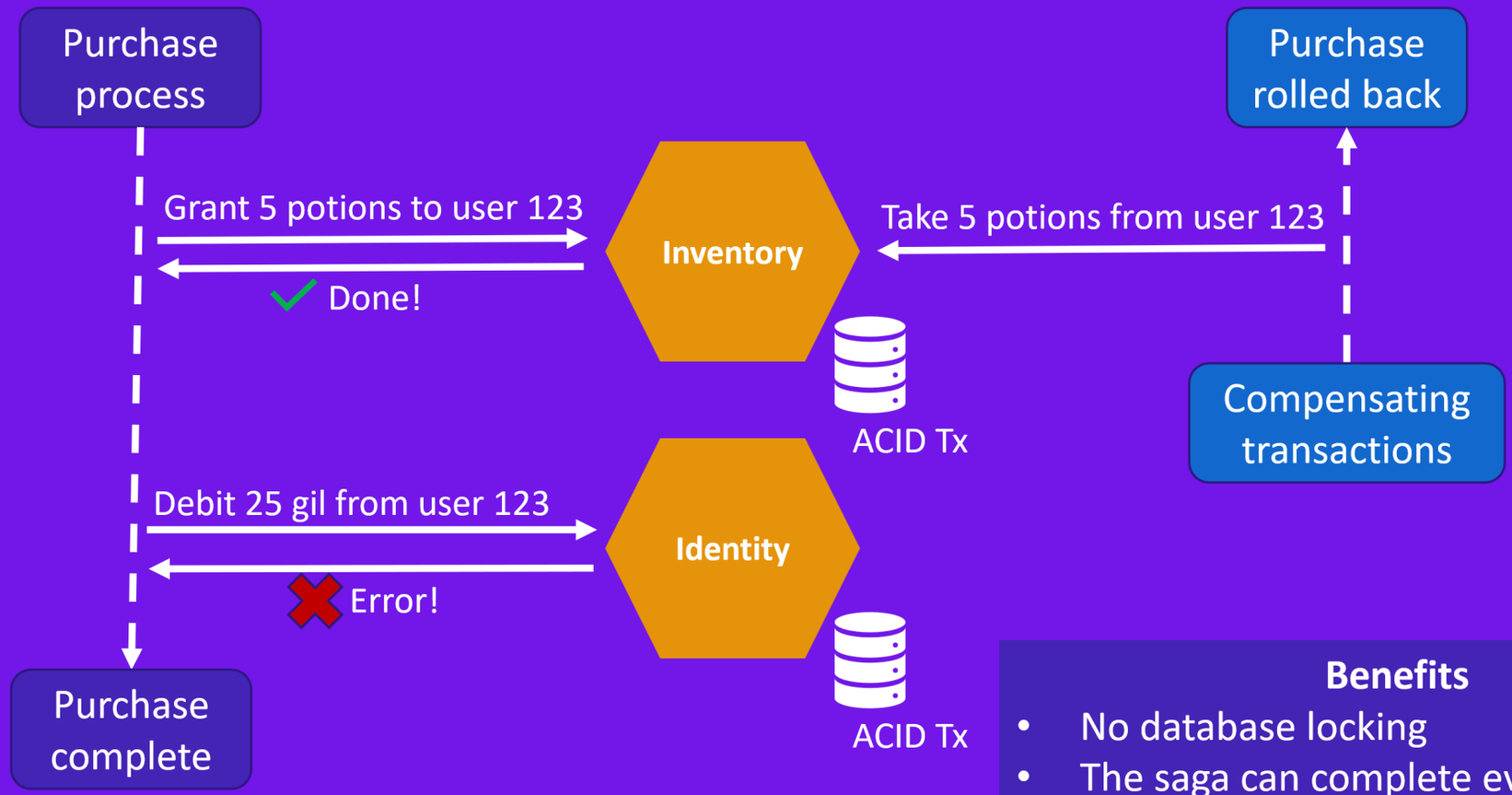
- Can inject huge amounts of latency into the system
- The database locks can become a performance bottleneck
- What if one of the microservices becomes unavailable in the commit phase?

What is a Saga?

"A Saga is a long lived transaction that can be written as a sequence of transactions that all complete successfully or the compensating transactions are executed to amend a partial execution."

SAGAS
by
Hector Garcia-Molina and Kenneth Salem
1987

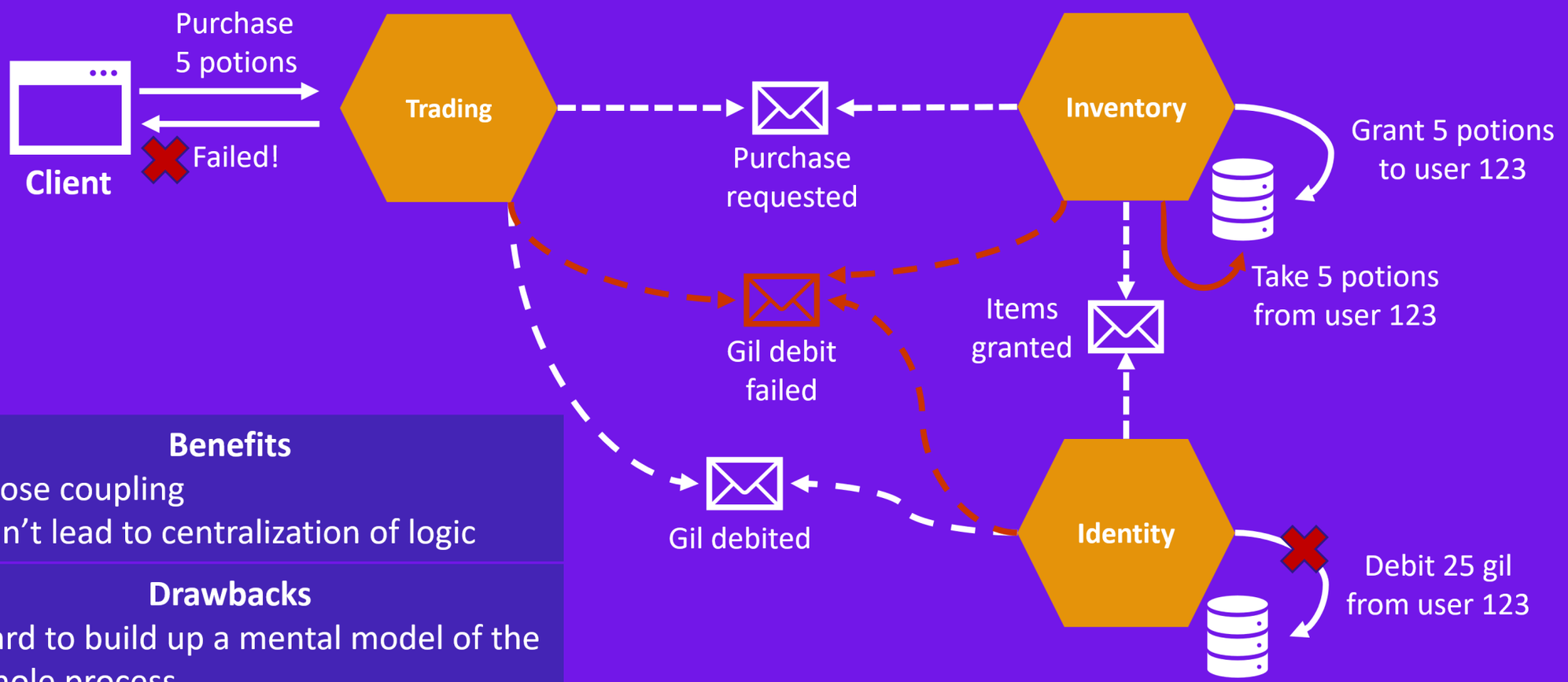
How do Sagas work?



Benefits

- No database locking
- The saga can complete even if participants are temporarily unavailable

Choreography-based Saga



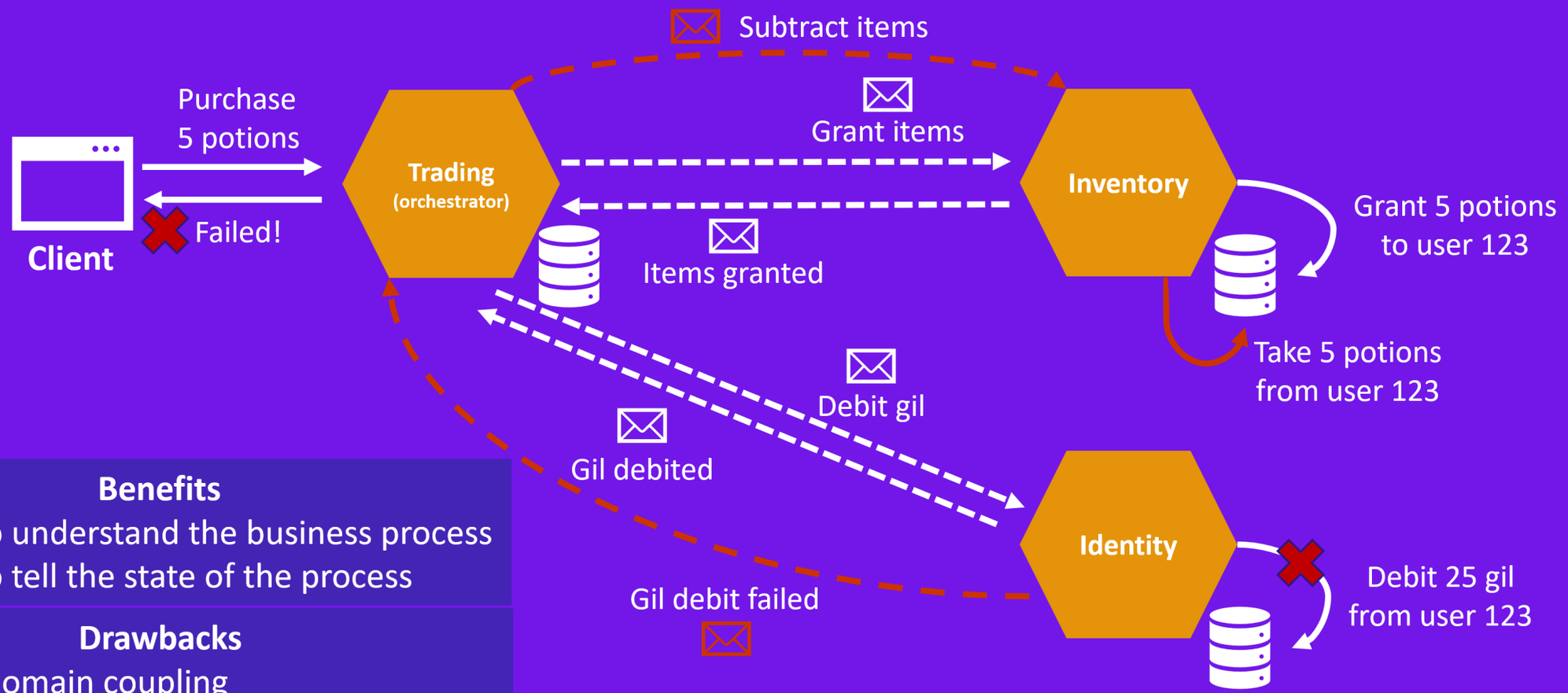
Benefits

- Loose coupling
- Can't lead to centralization of logic

Drawbacks

- Hard to build up a mental model of the whole process
- What is the current state of the saga?

Orchestration-based Saga



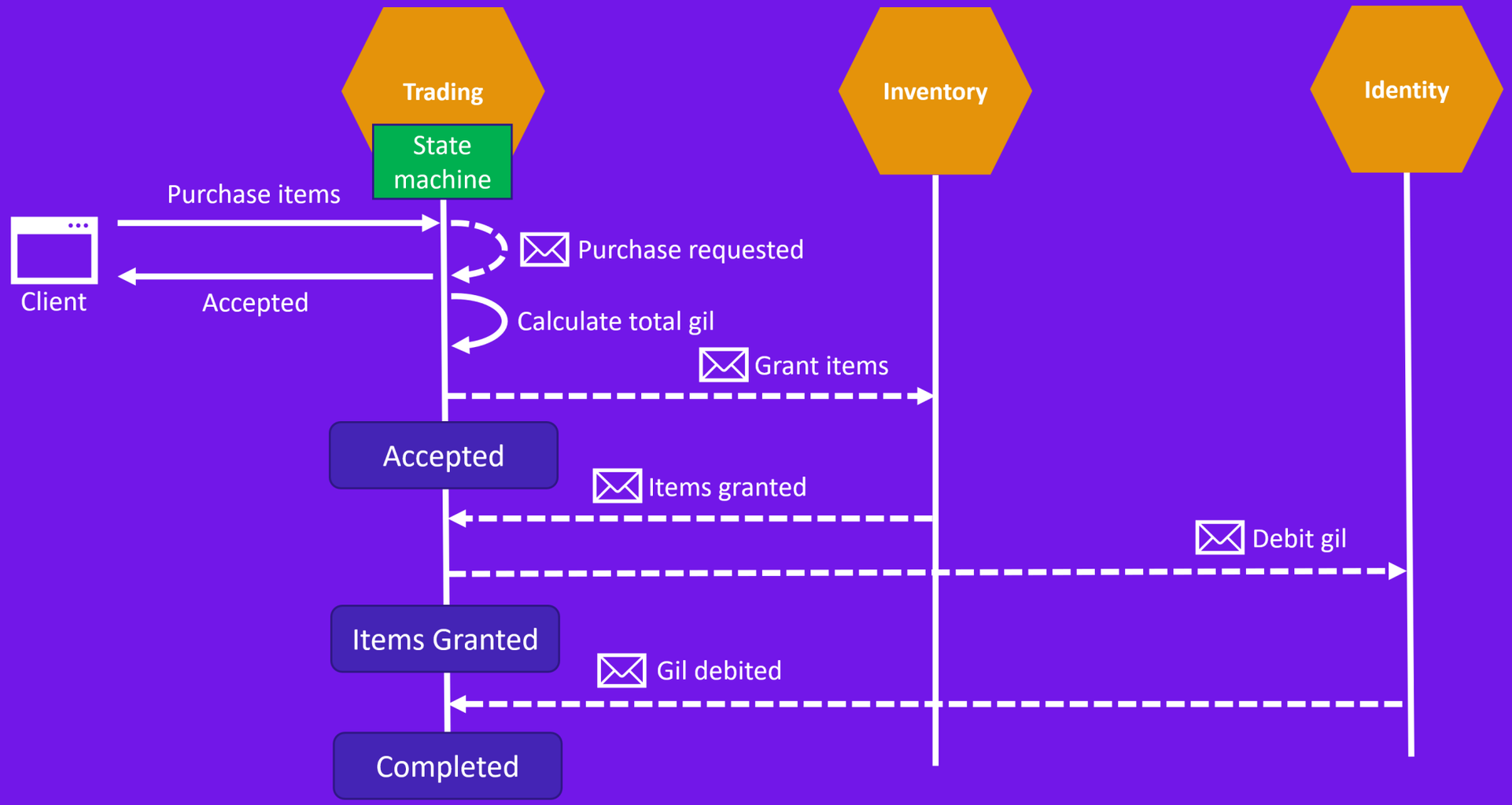
Benefits

- Easier to understand the business process
- Easier to tell the state of the process

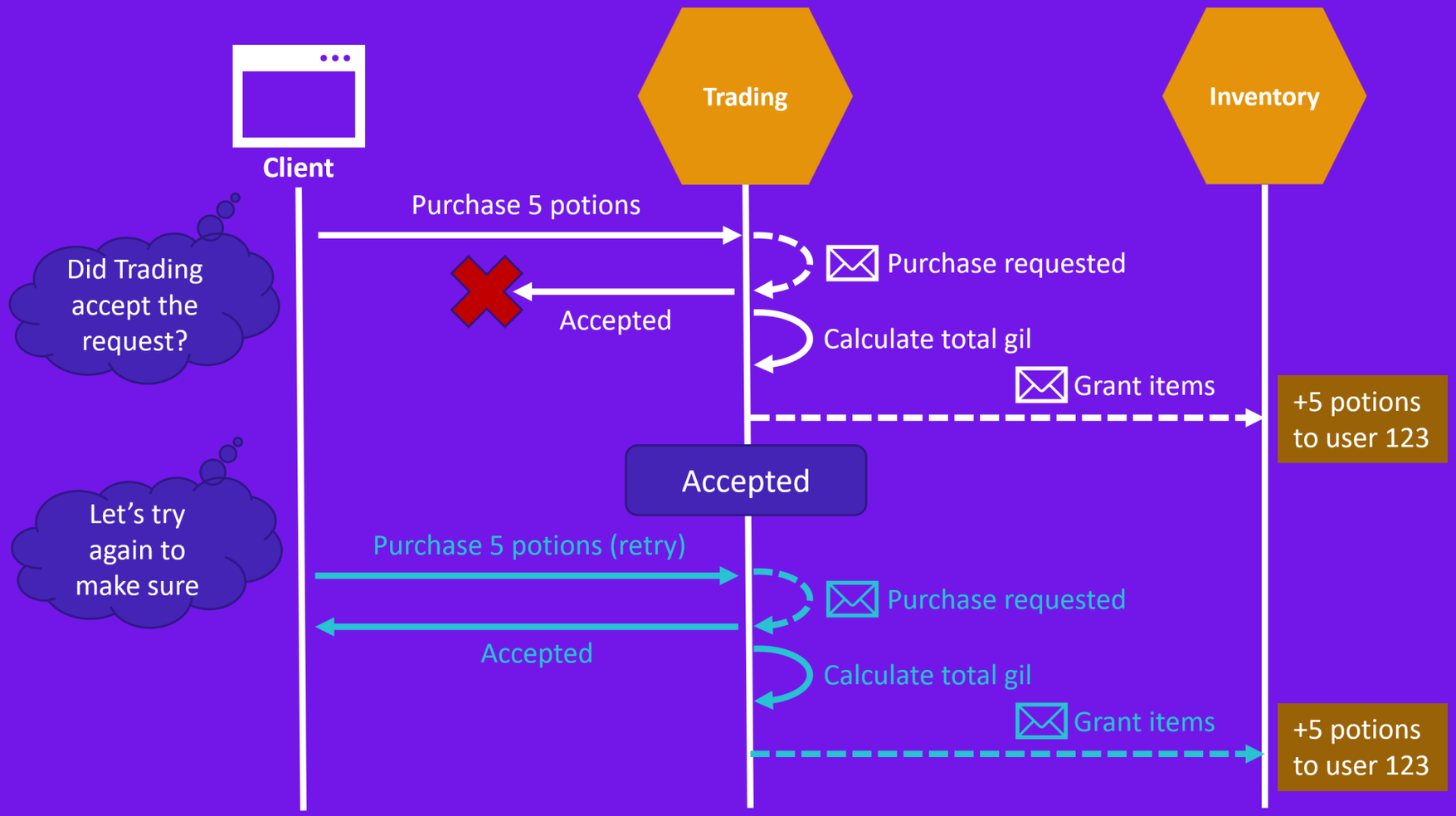
Drawbacks

- Higher domain coupling
- Risk of centralizing too much in the orchestrator

The purchase saga



The need for idempotency



Issues

- The client retried without caring about the outcome of the original request
- Trading has no idea the purchase is being retried
- Trading duplicates the purchase

What is idempotency?

The ability to apply the same operation multiple times without changing the result beyond the first try

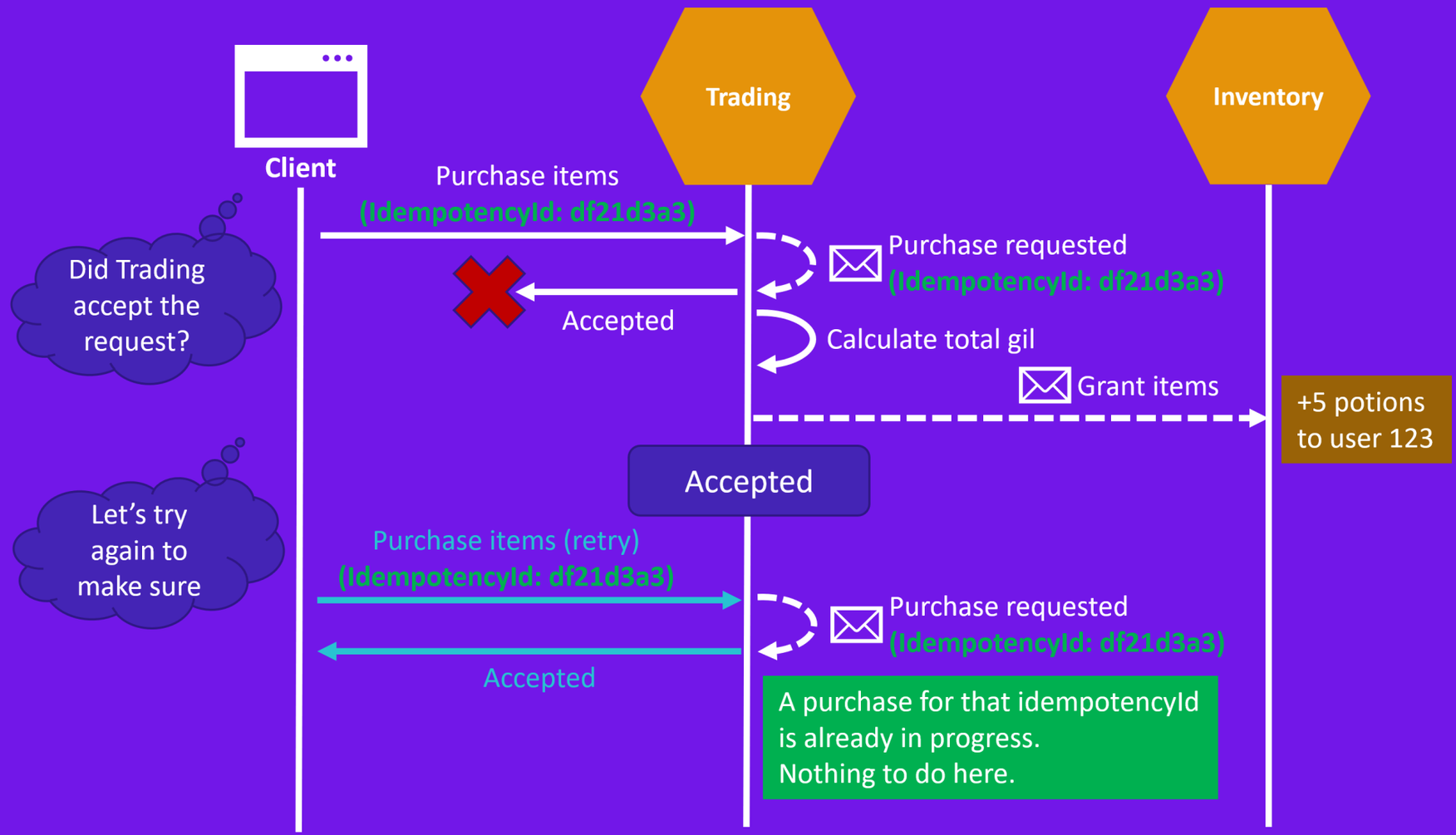
```
PUT /items/170928b3-7c55-4d54-abf1-d1a160826332
{
    "name": "Antidote",
    "price": 8
}
```

Naturally idempotent

```
POST /purchase
{
    "ItemId": "df21d3a3-35cb-41f5-81aa-a694f2168c98",
    "Quantity": 1
}
```

Not idempotent

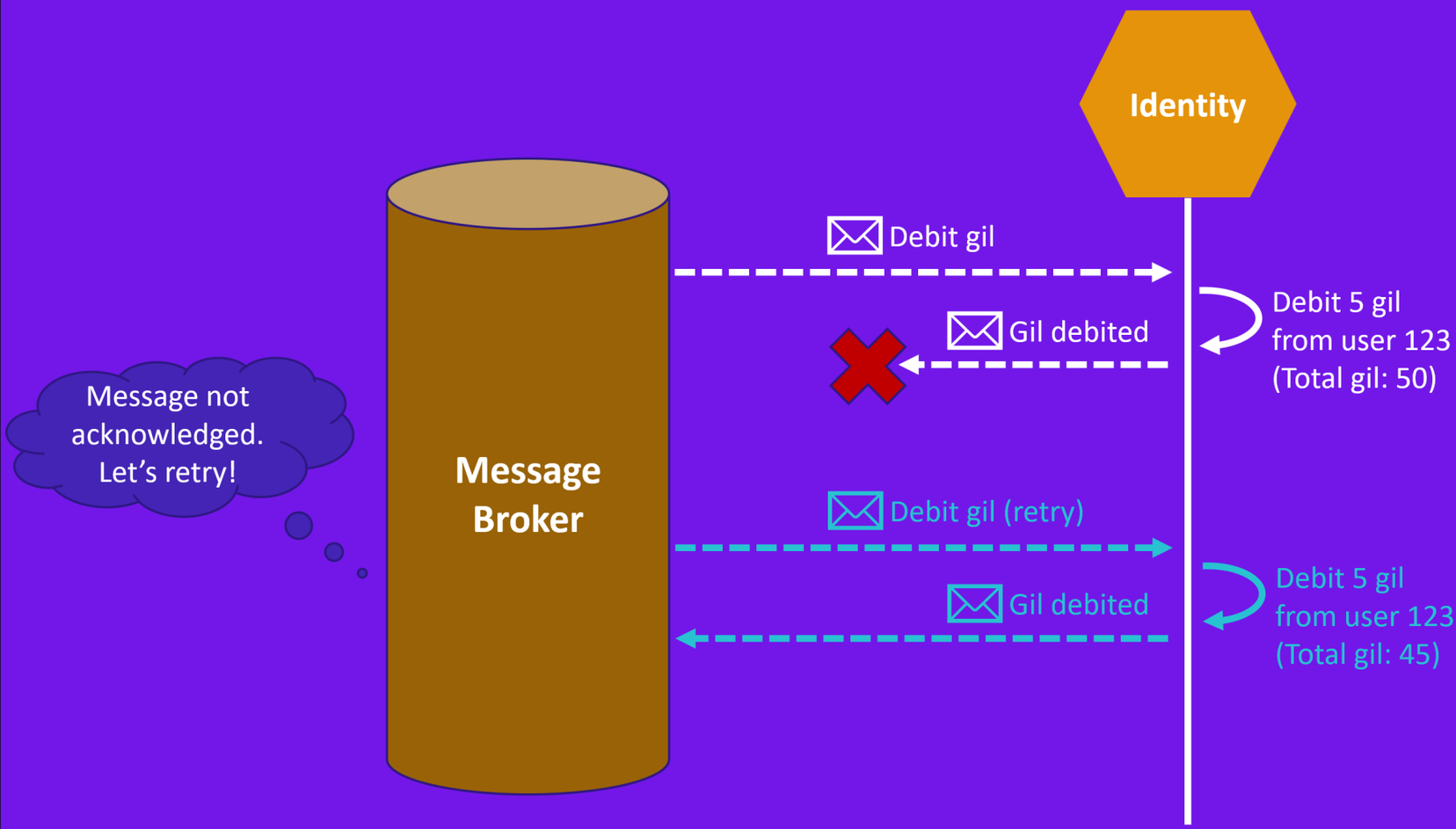
Adding idempotency



Benefits

- The client can safely retry the operation
- Trading can easily correlate and skip duplicated operations

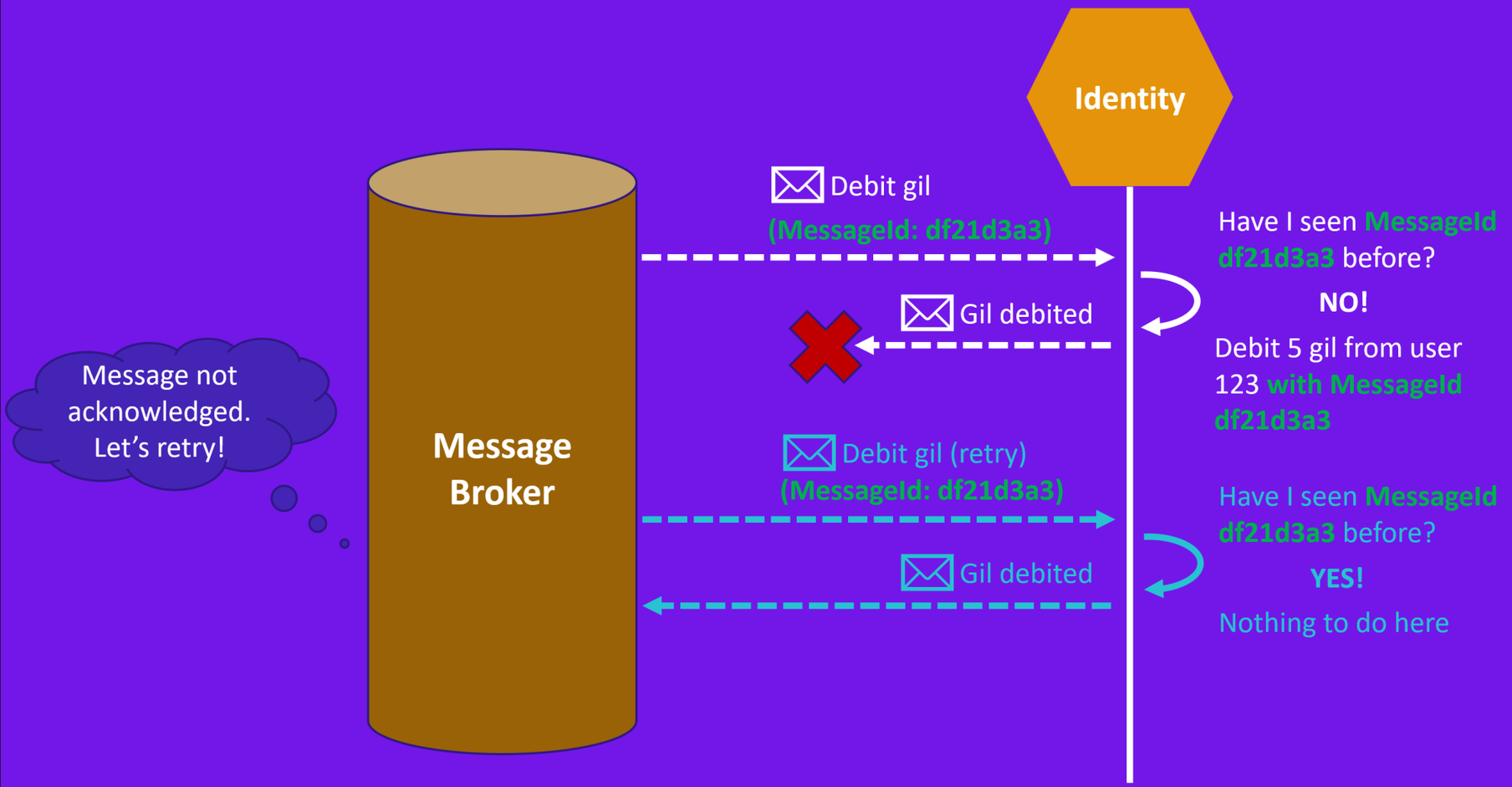
Message duplication



Issues

- The service consumes every message as if it is a new message
- The lack of idempotency results in an incorrect user's gil balance

Idempotent consumer



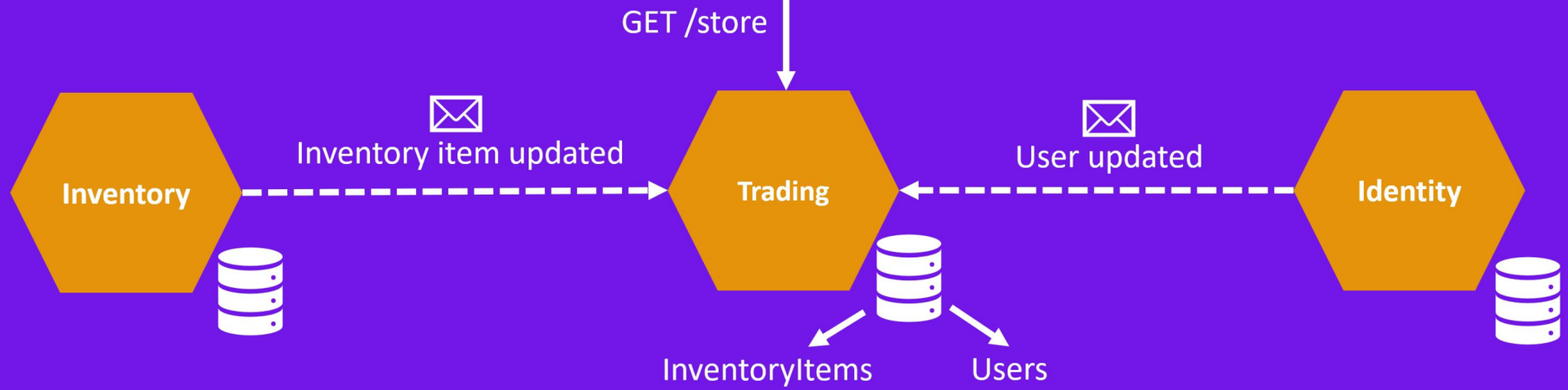
Benefits

- Processing the same message twice is the same as processing it once
- No inconsistencies in the user's gil balance

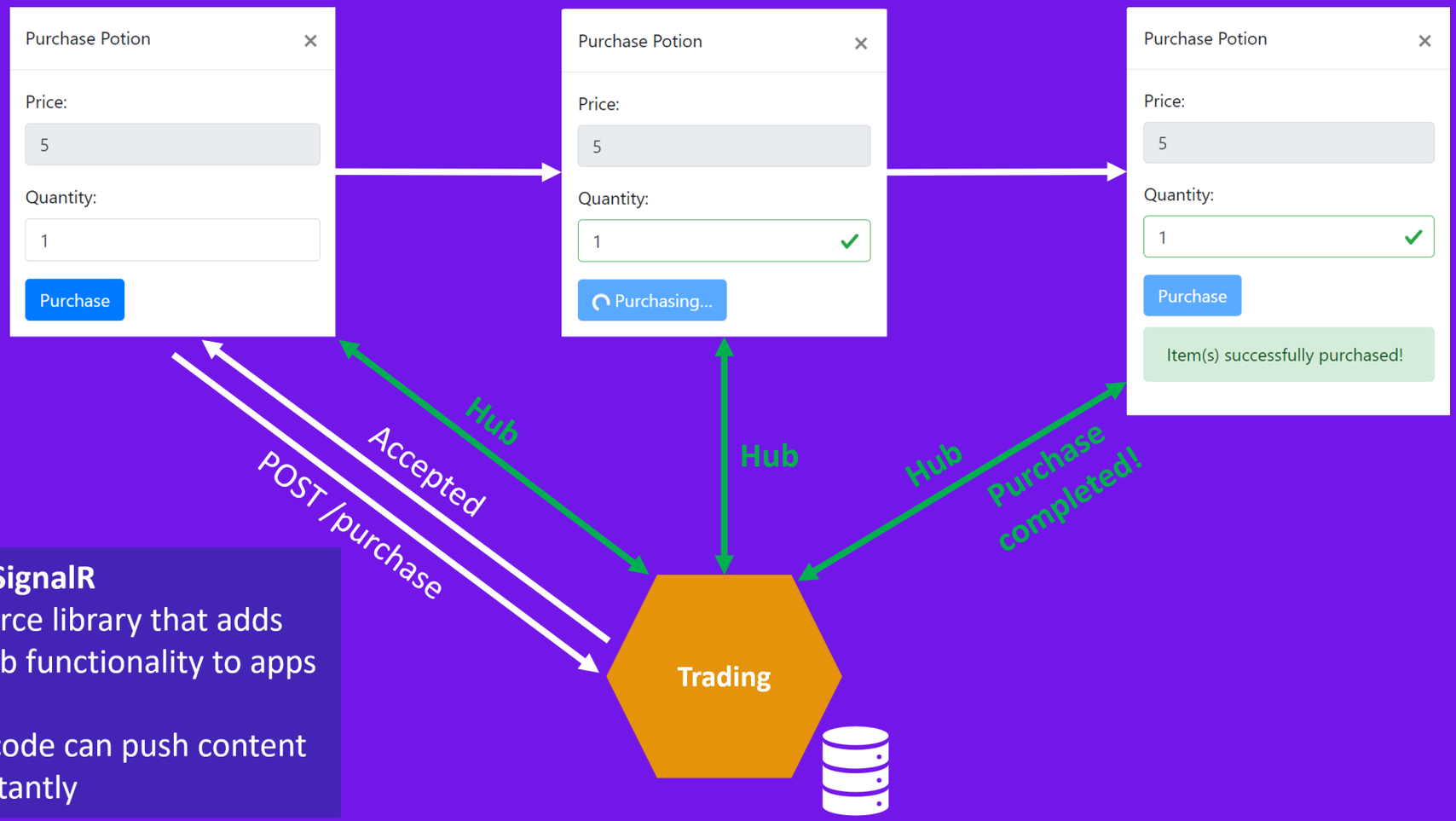
The store experience

My Gil: 67

Name	Description	Price	Owned	Actions
Potion	Restores some HP	5	3	Purchase
Antidote	Cures poison	8	0	Purchase
Ether	Restores a small amount of MP	9	2	Purchase



The purchase experience



SignalR

- An open-source library that adds real-time web functionality to apps
- Server-side code can push content to clients instantly