Design Decisions That Undermine API Security

Anas Mazioudi

@disklosr
github.com/disklosr
disklosr.com

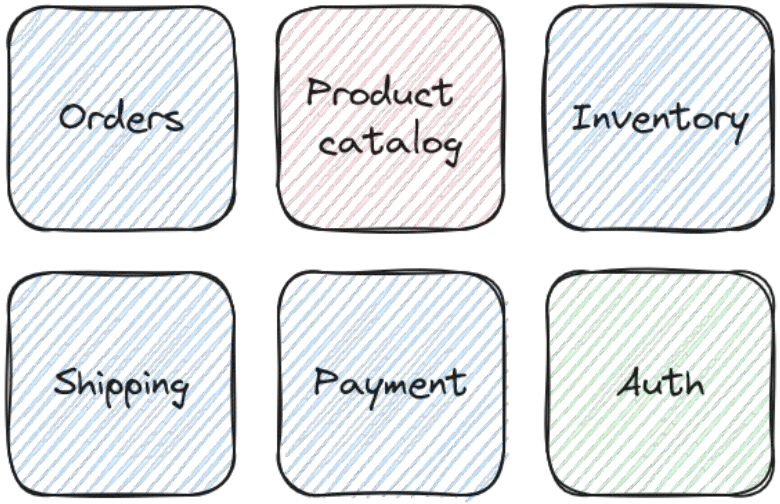
keywer.com

apisecuniversity.com

Tacoma narrows bridge

Setting up the stage

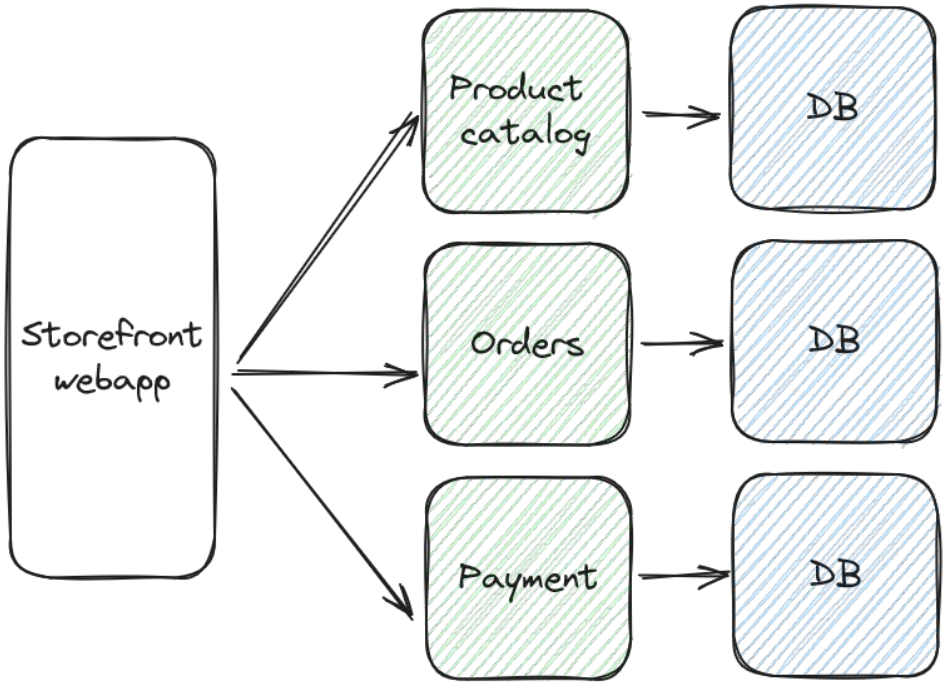
- Classic e-shop website
- Domain analysis completed
- 3 main design decisions
  - Architecture
  - Session management
  - Input validation



Architecture

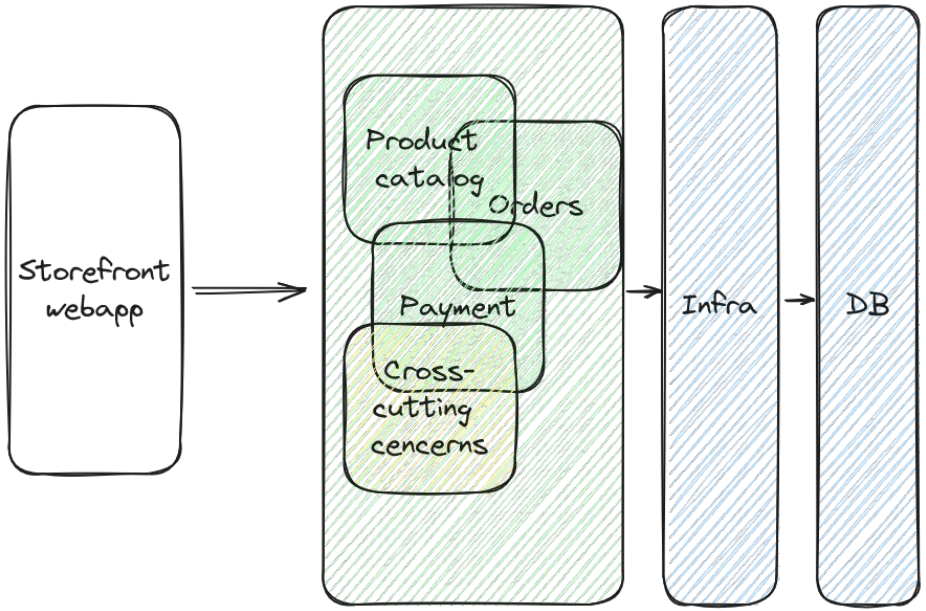
Microservices

- Small
- Independent
- Specialized
- Dedicated database
- Managed by different teams
- Possibly using different stacks
- Can be scaled individually



Monolith

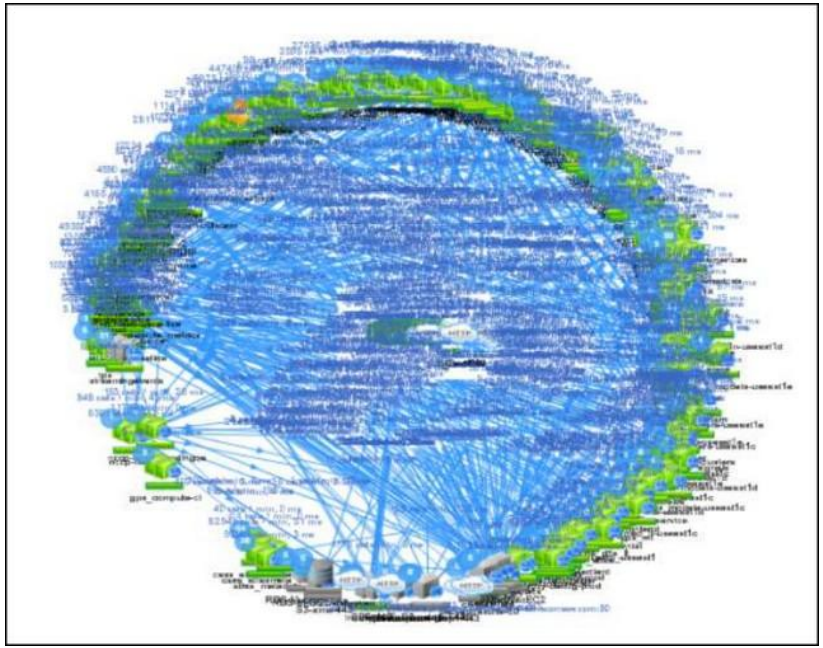
- Typical n-tier architecture
- One deployment unit
- Spaghetti business layer
- Tightly coupled concerns
- Bad reputation



Hidden cost of microservices

- Infrastructure complexity
- Operational overhead
- Performance
- Resilience
- Eventual consistency

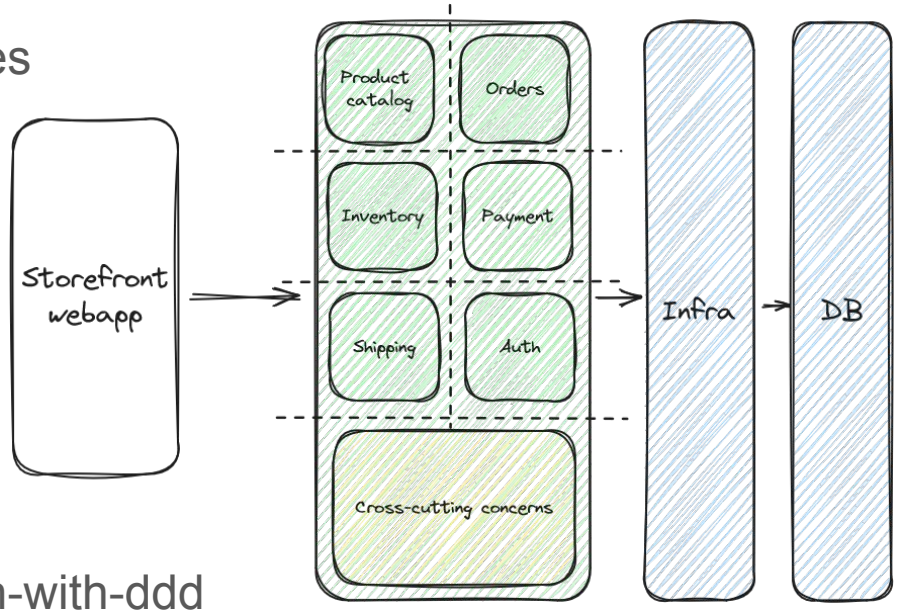
Complex = Harder to secure



Source: Netflix

Modular Monolith

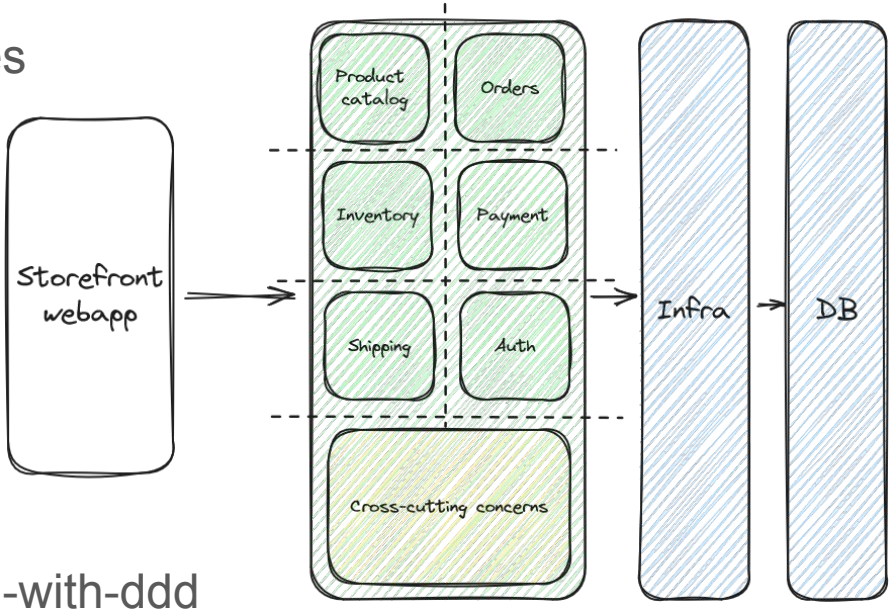
- Doing monoliths the right way
- Applying good old design principles
  - Independent modules
  - Clear boundaries
  - Loose coupling
  - Method call vs In-memory bus
- Single repo
- One deployment unit



github.com/kgrzybek/modular-monolith-with-ddd

Modular Monolith

- Doing monoliths the right way
- Applying good old design principles
  - Independent modules
  - Clear boundaries
  - Loose coupling
  - Method call vs In-memory bus
- Single repo
- One deployment unit



github.com/kgrzybek/modular-monolith-with-ddd

Case study

- Doctolib, major French SaaS provider in 3 major european countries
- Two web apps, for individuals and health professionals
- Over 300 developers
- 3 prod releases per day
- Peak activity @10M req/mn on jul'12 2021
- Over 2.5K servers

https://touilleur-express.fr/2023/07/12/un-monolithe-cest-quoi/

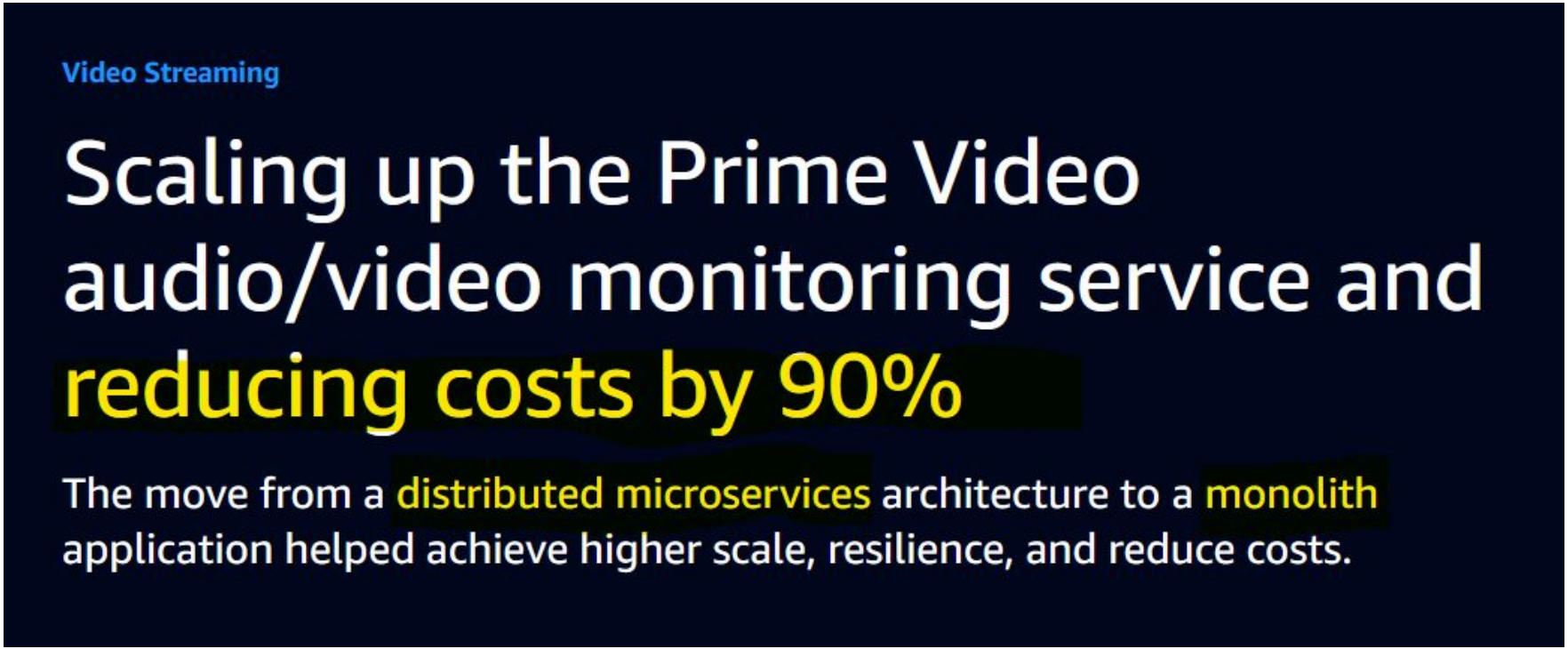
Case study

- Doctolib, major French SaaS provider in 3 major european countries
- Two web apps, for individuals and health professionals
- Over 300 developers
- 3 prod releases per day
- Peak activity @10M req/mn on jul'12 2021
- Over 2.5K servers

Using a monolith: mono-repo, single deployment unit, single database

https://touilleur-express.fr/2023/07/12/un-monolithe-cest-quoi/

Case study



https://www.primevideotech.com/video-streaming/scaling-up-the-prime-video-audio-video-monitoring-service-and-reducing-costs-by-90

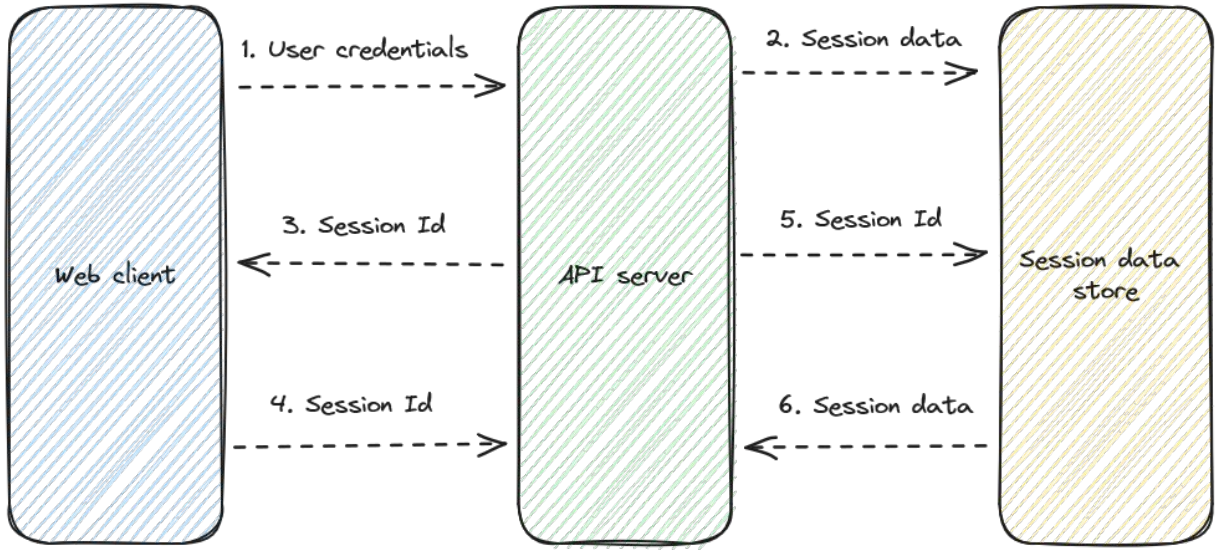
Takeaway #1

Microservices create more problems than they solve
Monoliths, when done right, can go a long way

Session management

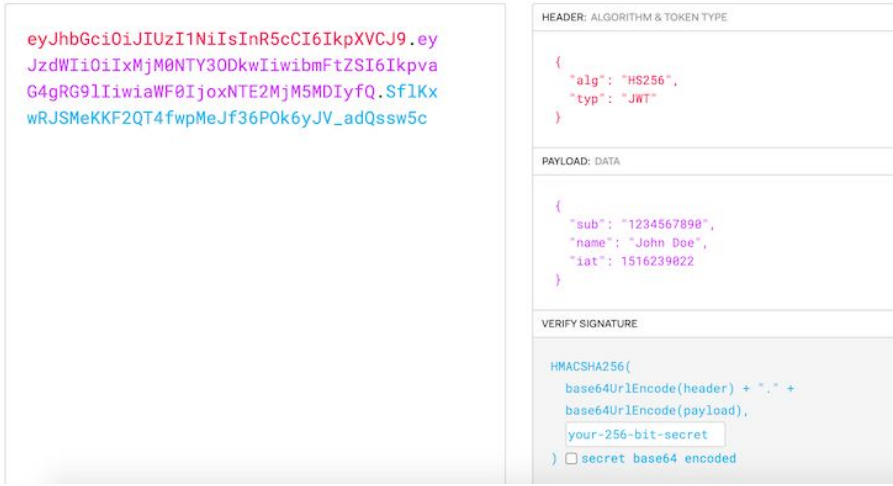
Session management

- Avoids asking user for credentials each time they interact with API
- Doesn’t apply for server-to-server API calls



Json Web Tokens (Jwt)

- A specification on how to securely store and exchange session data between parties in json format
- Composed of 3 parts
- Can hold arbitrary data called claims
- Saves db roundtrips



Session cookies

- Traditional way of doing session management
- Simple to understand
- Work out of the box
- Session data stored on the backend

Session management

What to choose?

- Cookies
  - Simple design
  - Secure if used with `httponly`, `secure`, `samesite=strict`
  - Nothing inherently bad about them
- Jwt
  - Easy to make mistakes when validating manually
  - Requires good understanding to use them right
  - Payload data can get stale
  - Hard to revoke

OWASP API Top 10 (2023): Broken Authentication

An API is vulnerable if it:

- Permits credential stuffing where the attacker uses brute force with a list of valid usernames and passwords.
- Permits attackers to perform a brute force attack on the same user account, without presenting captcha/account lockout mechanism.
- Permits weak passwords.
- Sends sensitive authentication details, such as auth tokens and passwords in the URL.
- Allows users to change their email address, current password, or do any other sensitive operations without asking for password confirmation.
- Doesn't validate the authenticity of tokens.
- Accepts unsigned/weakly signed JWT tokens (`{"alg":"none"}`)
- Doesn't validate the JWT expiration date.
- Uses plain text, non-encrypted, or weakly hashed passwords.
- Uses weak encryption keys.

Session management

What to choose?

- Cookies
  - Simple design
  - Nothing inherently bad about them
  - Battle-tested
  - Secure if used with `httponly`, `secure`, `samesite=strict`
- Jwt
  - Easy to make mistakes when validating
  - Requires good understanding to use them right
  - Payload data can get stale
  - Hard to revoke

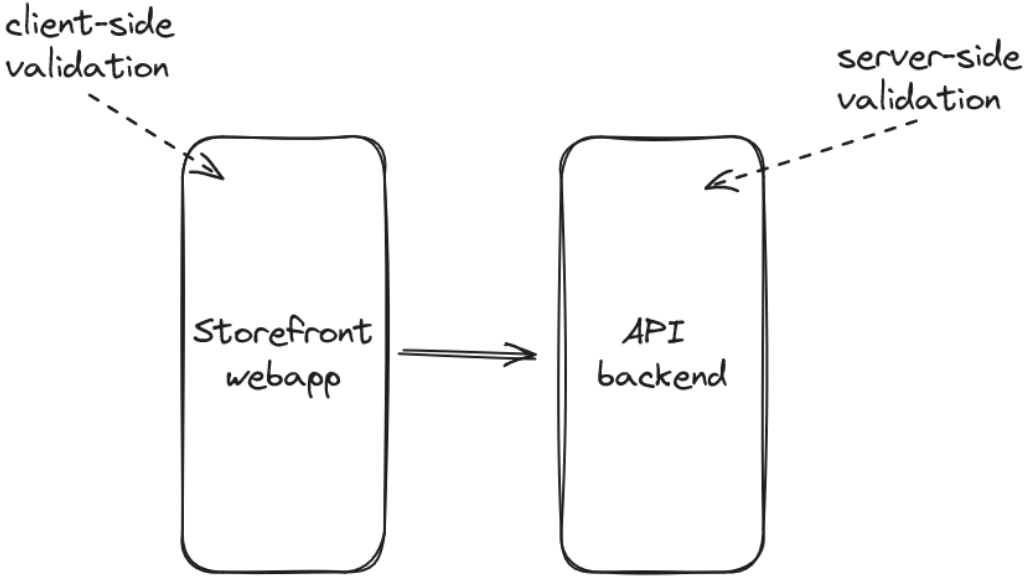
Takeaway #2

Don’t dismiss using cookie-based session management
They’re still relevant and provide simplicity and great security

Input validation

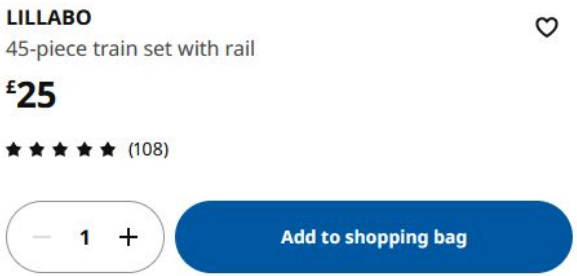
Input validation

- Ensure data sent by clients adheres to defined constraints
- Important security measure
- Can be implemented at different levels



Client-side validation

- A convenience for users
- Saves time by avoiding unnecessary round trips to servers when the client is sure the request would fail server validation anyway
- You know it when someone is messing with your API



app prevents adding an item with 0 quantity to cart

Server side validation

- An crucial security measure to prevent API misuse
  - Injection attacks
  - Data corruption
- Reports back to callers why validation failed
- Usually implemented at controller level
- Should be used to validate the shape and types of received data

Server side validation

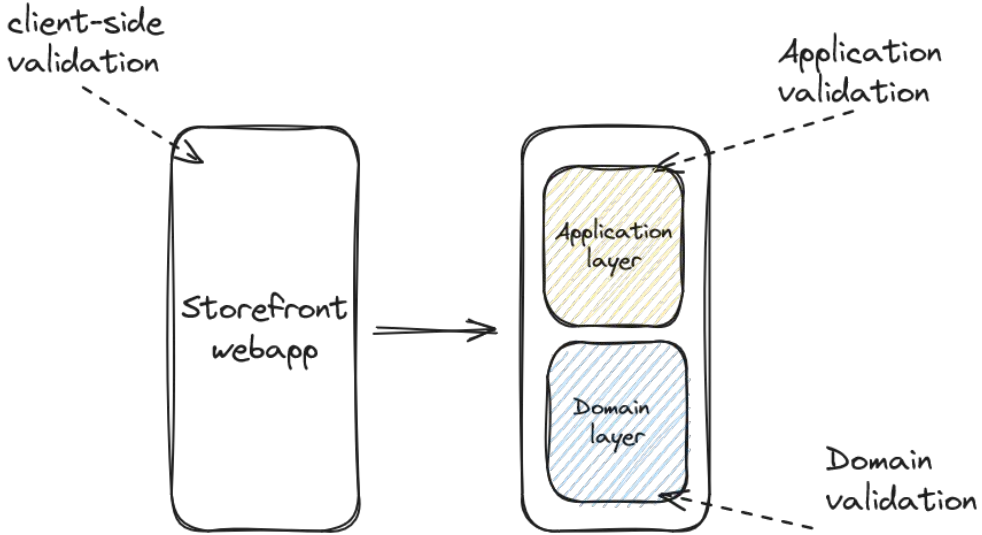
```
public class BasketItemDTO
{
    public int ProductId { get; set; }
    public string ProductName {get; set;}
    public int Quantity { get; set; }
}

public class BasketItemDTOValidator
{
  public BasketItemDTOValidator()
  {
    RuleFor(x => x.ProductId).GreaterThan(0);
    RuleFor(x => x.ProductName).NotEmpty().Length(1,250);
    RuleFor(x => x.Quantity).InclusiveBetween(1,10);
  }
}
```

Server side validation

- An crucial security measure to prevent API misuse
  - Injection attacks
  - Data corruption
- Usually implemented at controller level
- Should be used to validate the shape and types of received data
- **Not a substitute for domain validation**

Domain validation

- Implements business rules
- Enforces domain invariants
- Allows an easy way to test business rules

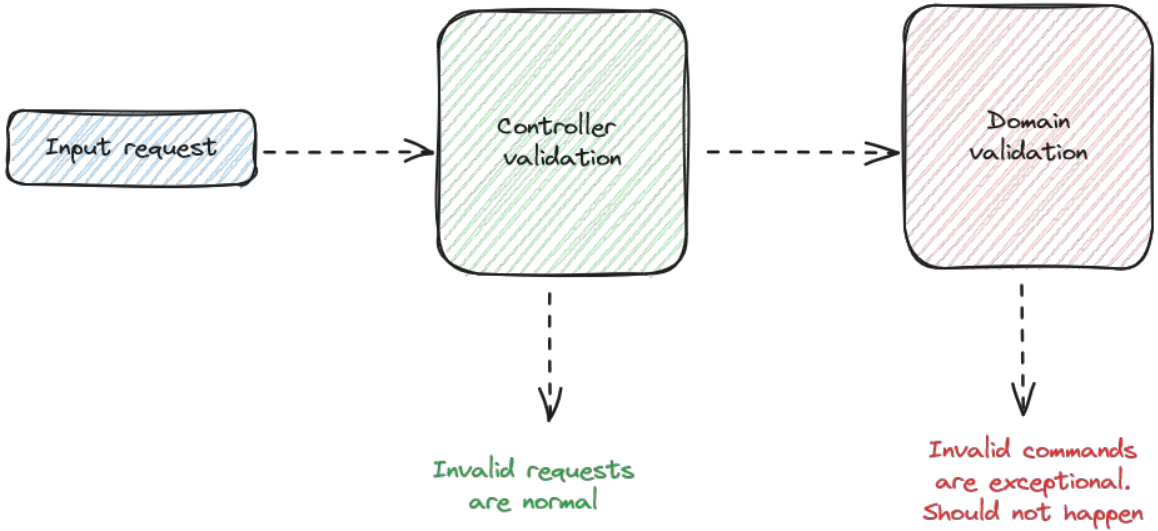


Domain validation

```
// Domain layer
public class BasketItem
{
    public BasketItem(int productId, string productName, int quantity)
    {
        if (quantity < 1)
            throw new DomainException("Basket item quantity cannot be < 1")

        ...
    }
}
```

Server side validation

- Failing controller validation is not an error
- Failing domain validation is a business rule violation = Error
- Domain validation acts as a fortification line in case the first line of defense fails



Takeaway #3

Implement proper validation on both client and server side.
Use your domain model to enforce business invariants and fail when incorrect commands are detected

Wrapping up

- Resist the temptation to use micro-services unless it’s really justified
  - Complexity can weaken the security of the system
- You can use session-based cookies for session management in your APIs
  - Long lived Jwt can be a security issue and give access to unauthorized users
- Implement proper validation at client and server-side
  - Enforcing business rules and invariants at domain level can be a good second line of defense in case the first one fails

Thank you

Anas Mazioudi

@disklosr
github.com/disklosr
disklosr.com

keywer.com

apisecuniversity.com