

Pattern-uri Arhitecturale

Stefan Turcu

turcu.98.stefanel@gmail.com

Decembrie 2024



It depends!

Răspunsul la toate întrebările despre programare

Arhitectura CQRS - *Command Query Responsibility Segregation*

Optimizarea aplicațiilor prin separarea responsabilităților

Ce este CQS? - command query separation

Definiție: Principiu propus de Bertrand Meyer care spune că o metodă trebuie să fie fie:

- **Command:** Execută o acțiune și modifică starea sistemului.
- **Query:** Returnează un rezultat fără a modifica starea.

CQS este un principiu simplu, dar puternic, pentru crearea unui cod ușor de întreținut și testat.

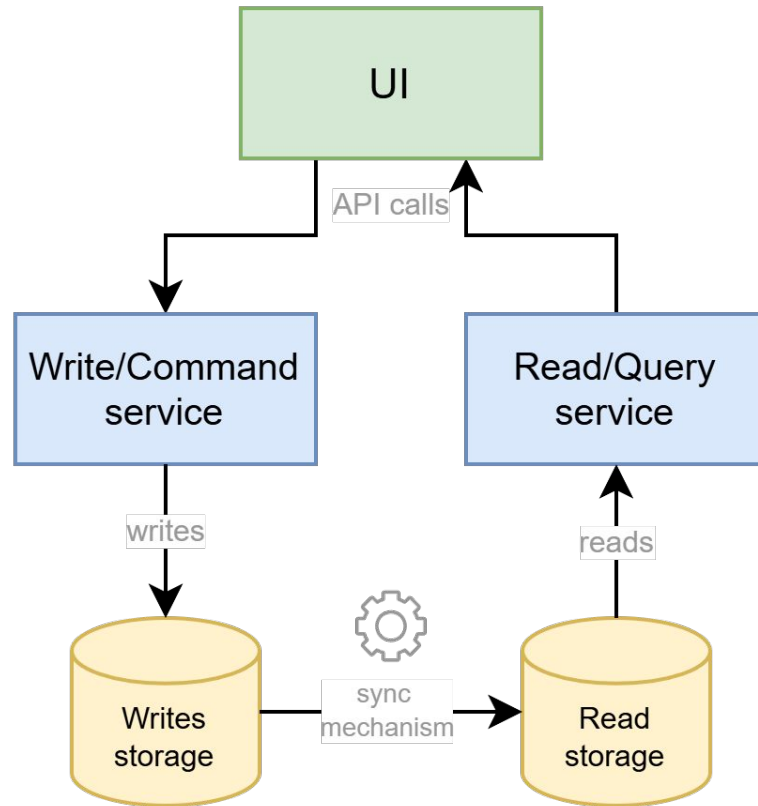
```
public class BankAccount {  
    public void Deposit(decimal amount); // Command  
    public decimal GetBalance();         // Query  
}
```

Ce este CQRS?

Definiție: Separarea responsabilităților între două *modele* distincte:

- **Modelul Command:** gestionează modificările (scrierea datelor).
- **Modelul Query:** gestionează interogările (citirea datelor).

CQRS permite proiectarea optimă a ambelor modele pentru nevoile lor specifice.



Avantajele CQRS

1. **Scalabilitate:** Modelele de citire și scriere pot fi scalate independent.
2. **Performanță:** Se pot optimiza interogările și modificările fără compromisuri.
3. **Claritate:** Responsabilitățile sunt clar separate.
4. **Flexibilitate:** Permite utilizarea mai multor tehnologii pentru citire și scriere.
5. **Evoluție controlată:** Ușor de adaptat pentru schimbări funcționale.

Dezavantajele CQRS

1. **Complexitate:** Introduce logică suplimentară și necesită sincronizare între modele.
2. **Consistență eventuală:** În aplicațiile distribuite, datele de citire pot fi învechite temporar.
3. **Costuri ridicate:** Mai multe baze de date sau componente pot crește costurile operaționale.
4. **Curba de învățare:** Necesită o înțelegere profundă pentru implementare corectă.

Principiile CQRS

1. **Separarea** strictă între citire și scriere.
2. **Independența modelelor:** Fiecare *model* poate fi proiectat pentru scopul său specific.
3. **Eventual Consistency:** Datele de citire pot reflecta modificările cu o întârziere acceptabilă.
4. **Mediatori:** Interacțiunile dintre componente sunt gestionate de mediere.

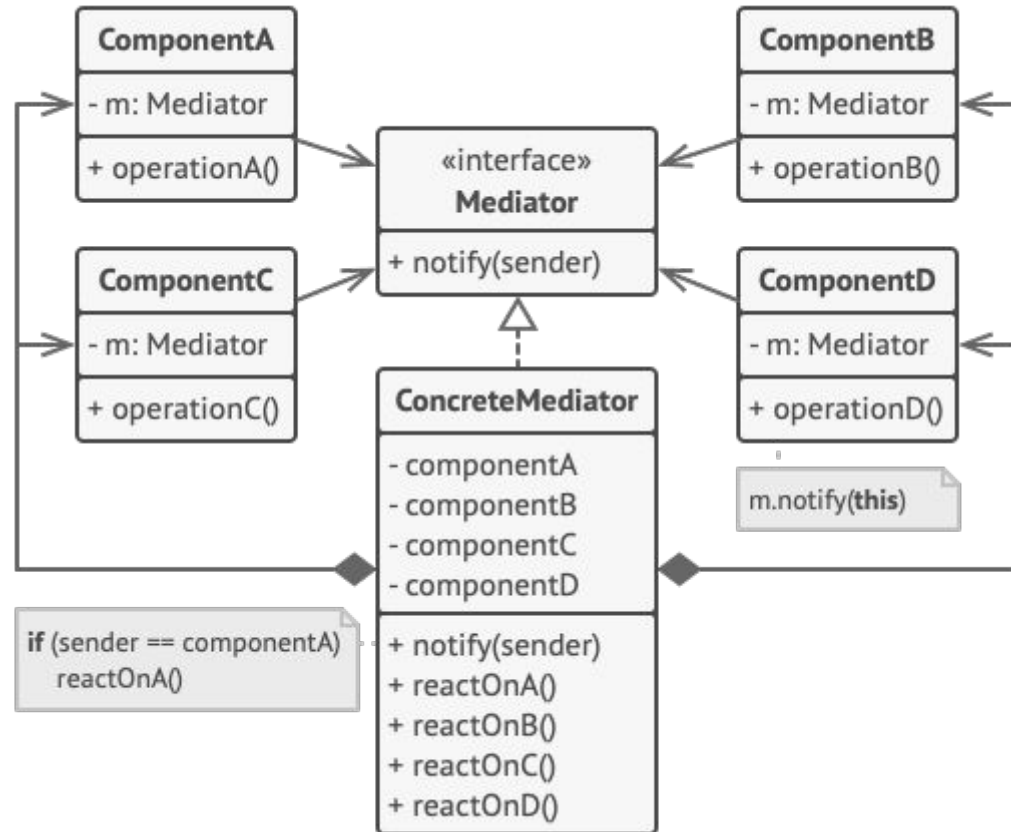
Când poate fi CQRS soluția potrivită?

Exemple de aplicații unde ar putea fi implementat

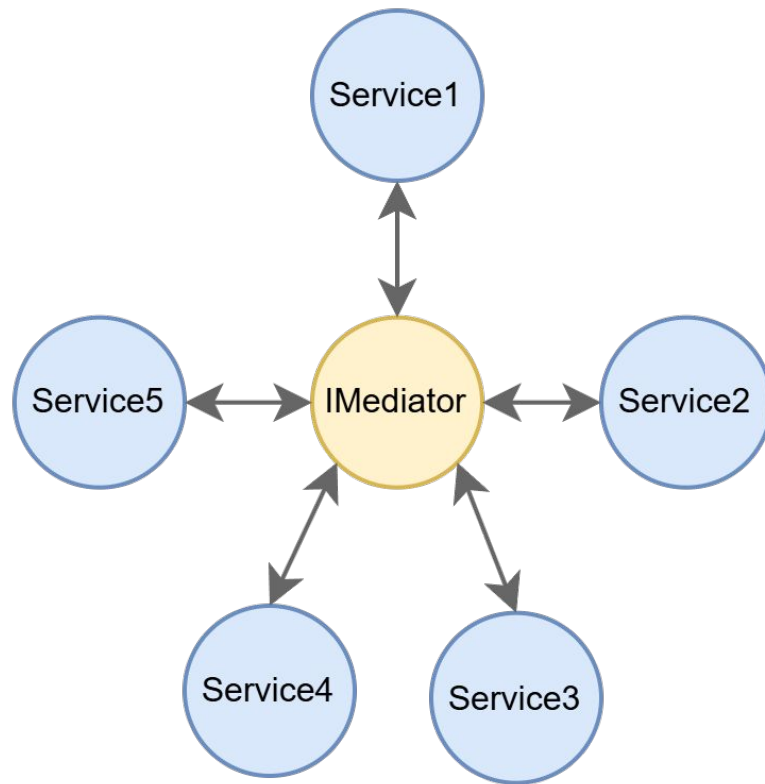
Mediator Pattern

Definiție:


Mediator Pattern este un model de design comportamental care facilitează comunicarea între componente (obiecte sau clase) printr-un mediator central, reducând dependențele directe dintre ele.



Libreria MediatR (C#)



Libreria MediatR (C#)

```
public class CreateOrderCommand : IRequest<bool>
{
     1 usage
    public string OrderName { get; set; }
}

public class CreateOrderHandler : IRequestHandler<CreateOrderCommand, bool>
{
    public async Task<bool> Handle(
        CreateOrderCommand request,
        CancellationToken cancellationToken
    )
    {
        // Logica de creare a comenzi
        Console.WriteLine($"Order {request.OrderName} created!");
        return true;
    }
}
```

Libreria MediatR (C#)

```
public class OrderController : ControllerBase
{
    private readonly IMediator _mediator;

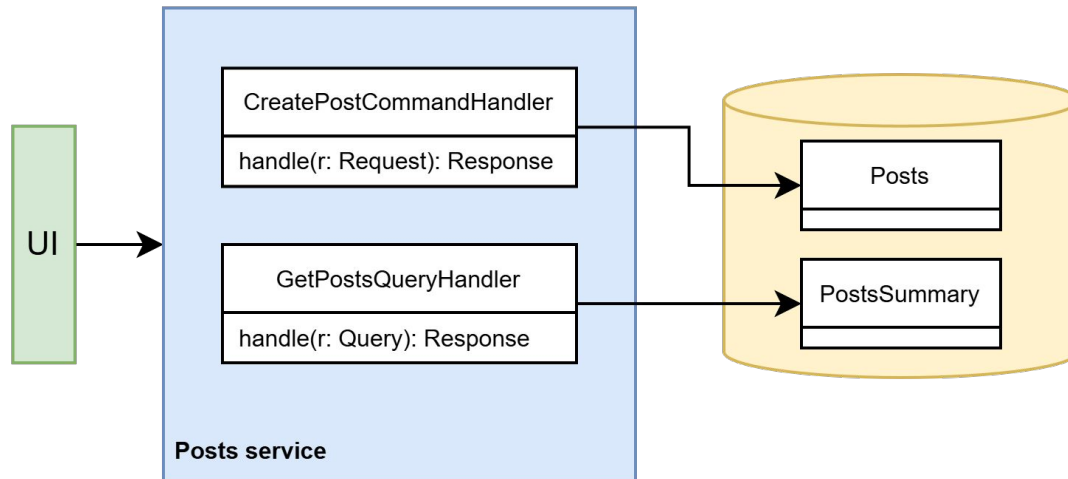
    public OrderController(IMediator mediator)
    {
        _mediator = mediator;
    }

    [HttpPost]
    public async Task<IActionResult> CreateOrder([FromBody] CreateOrderRequest request)
    {
        bool result = await _mediator.Send(request: new CreateOrderCommand { OrderName = request.OrderName });
        return result ?
            Ok("Order created successfully") :
            BadRequest(error: "Failed to create order");
    }
}
```

Implementari CQRS

CQRS într-un singur serviciu, utilizând o singură bază de date.

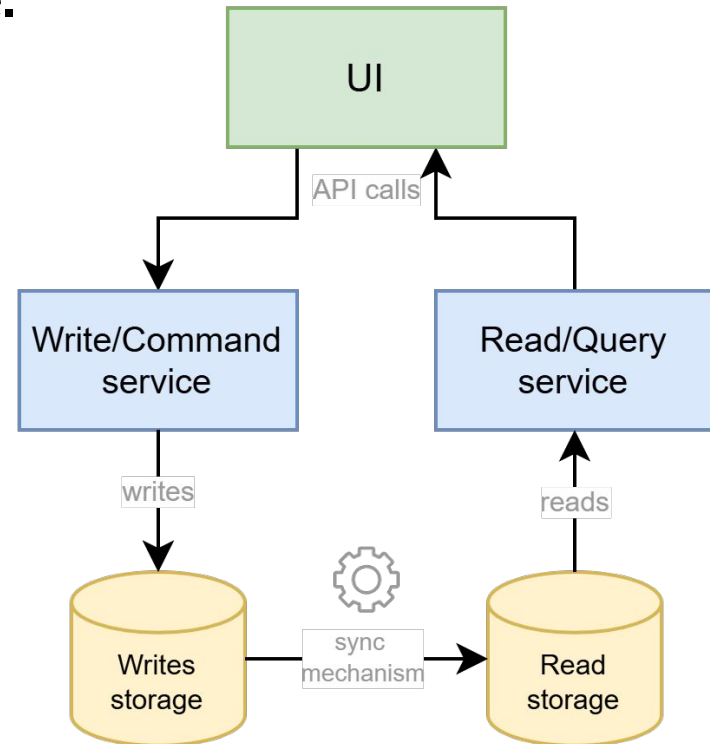
- Separarea logică a operațiunilor de citire (**queries**) și scriere (**commands**) prin handler-uri distincte.
- **Două tabele** (scriere și citire) pentru performanță sporită și separare logică.



Implementari CQRS

CQRS implementat prin separarea completă a operațiunilor de citire și scriere în două servicii independente.

- Fiecare serviciu are propria bază de date.
- **Command Service:** Gestionează operațiunile de scriere și păstrează date normalizate pentru consistență.
- **Query Service:** Gestionează operațiunile de citire și folosește date optimizate pentru interogări (denormalizate).



Implementari CQRS

CQRS implementat prin separarea completă a operațiunilor de citire și scriere în două servicii independente și folosind Event Sourcing

- **Event Sourcing** este un model arhitectural în care starea sistemului este determinată prin **rejucarea** unei secvențe de evenimente imutabile.
- Fiecare eveniment reprezintă o modificare de stare care a avut loc în sistem, stocată cronologic într-un **Event Store** (tabela).

PostsEventStore
EventId: number (PK)
EventType: varchar(255)
AggregateId: number
EventData: jsonb
CreatedAt: timestamp

Posts
PostId: number (PK)
Title: varchar(100)
Content: varchar(500)
Author: varchar(100)
CreatedAt: timestamp
LastUpdatedAt: timestamp

Implementari CQRS

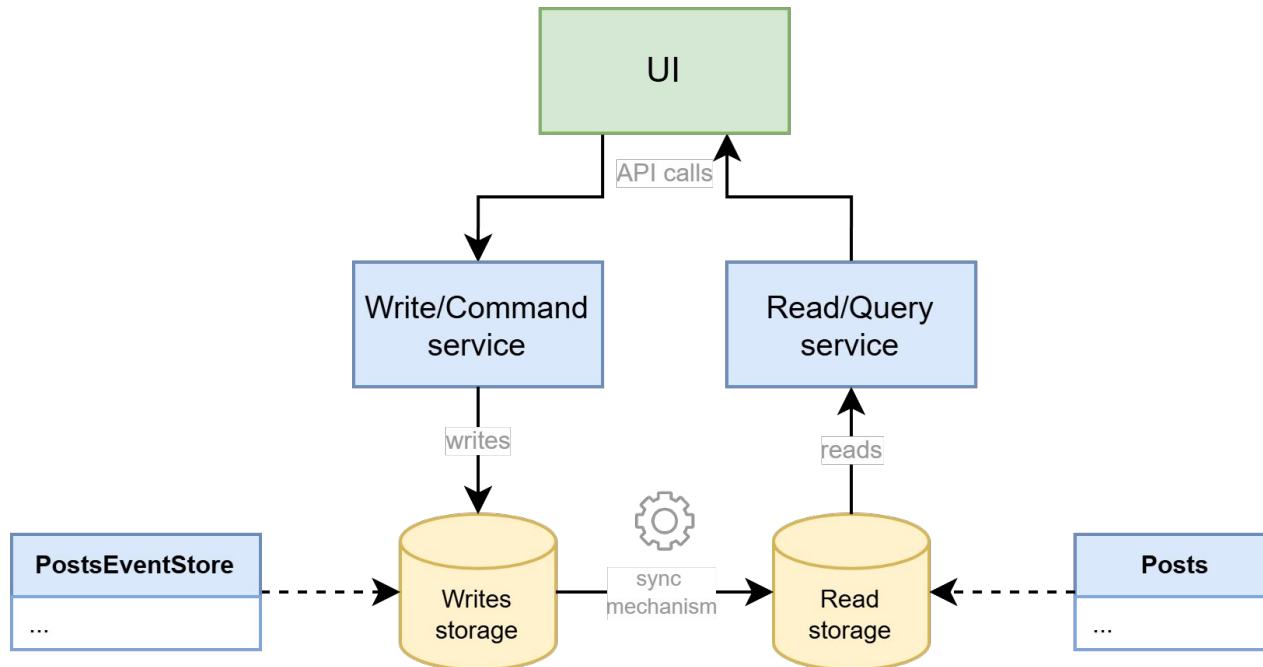
CQRS implementat prin separarea completă a operațiunilor de citire și scriere în două servicii independente și folosind Event Sourcing

```
{
  "EventId": "65",
  "EventType": "PostCreated",
  "AggregateId": "1",
  "Data": {
    "PostId": "1",
    "Title": "Introducing Event Sourcing",
    "Content": "Event sourcing is powerful.",
    "Author": "John Doe",
    "CreatedAt": "2024-12-04T12:00:00Z"
  },
  "Timestamp": "2024-12-04T12:00:00Z"
}
```

```
{
  "EventId": "66",
  "EventType": "PostUpdated",
  "AggregateId": "1",
  "Data": {
    "PostId": "1",
    "UpdatedFields": {
      "Title": "Understanding Event Sourcing",
    },
    "UpdatedAt": "2024-12-05T09:30:00Z"
  },
  "Timestamp": "2024-12-05T09:30:00Z"
}
```

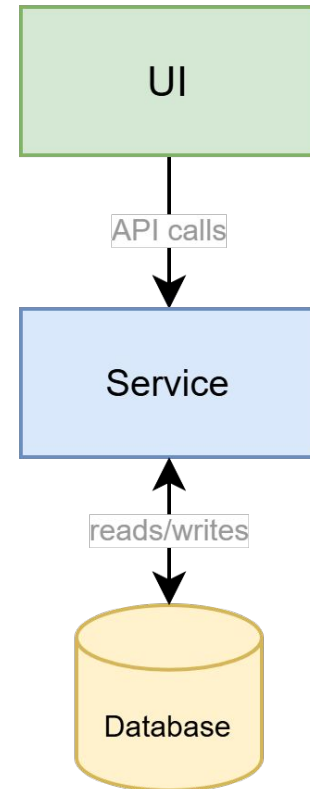
Implementari CQRS

CQRS implementat prin separarea completă a operațiunilor de citire și scriere în două servicii independente și folosind Event Sourcing

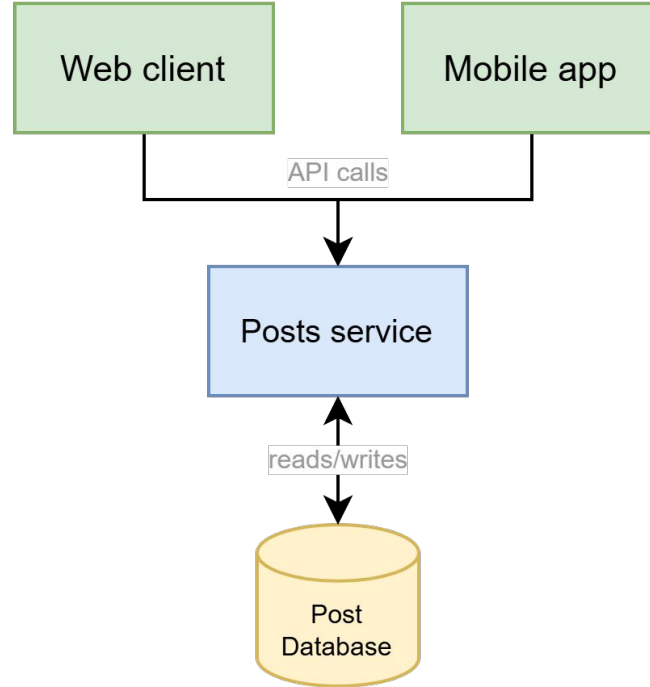


Probleme în Arhitecturile Tradiționale

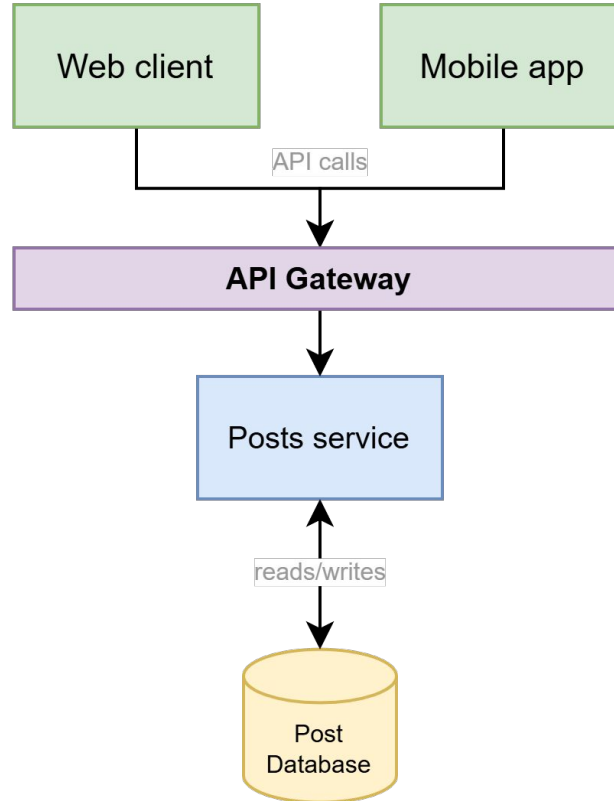
- **Un singur model pentru citire și scriere:** Același model de date gestionează atât citirea, cât și actualizarea datelor.
- **Blocaje de performanță:** Creșterea traficului duce la timpi de răspuns mai mari pe măsură ce aplicația se extinde.
- **Dificultăți în scalare:** Gestionarea simultană a operațiunilor mari de citire și scriere devine tot mai complexă.



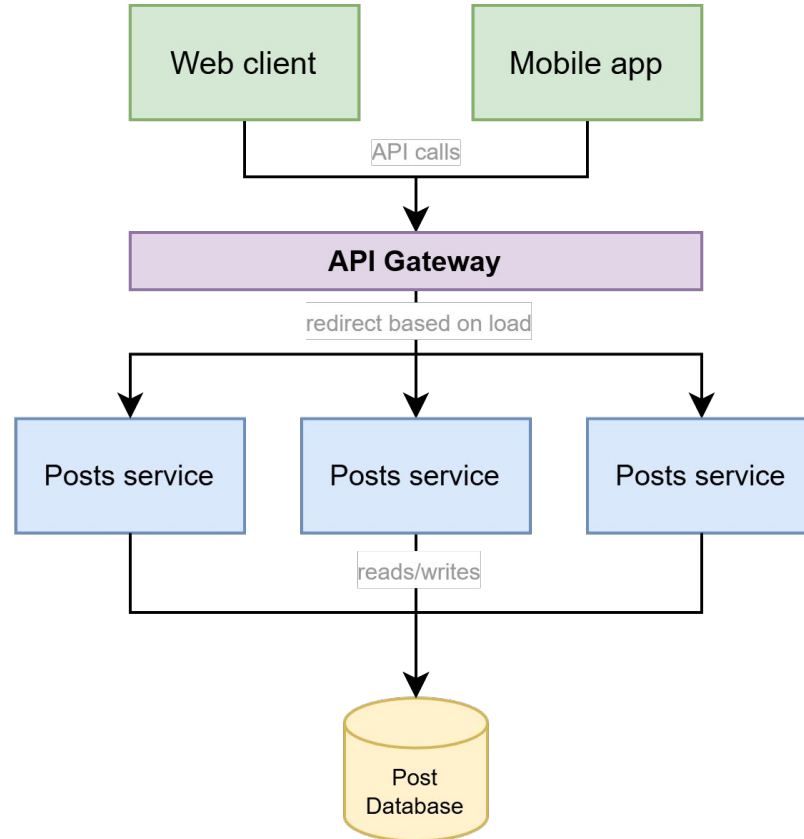
My Social Media



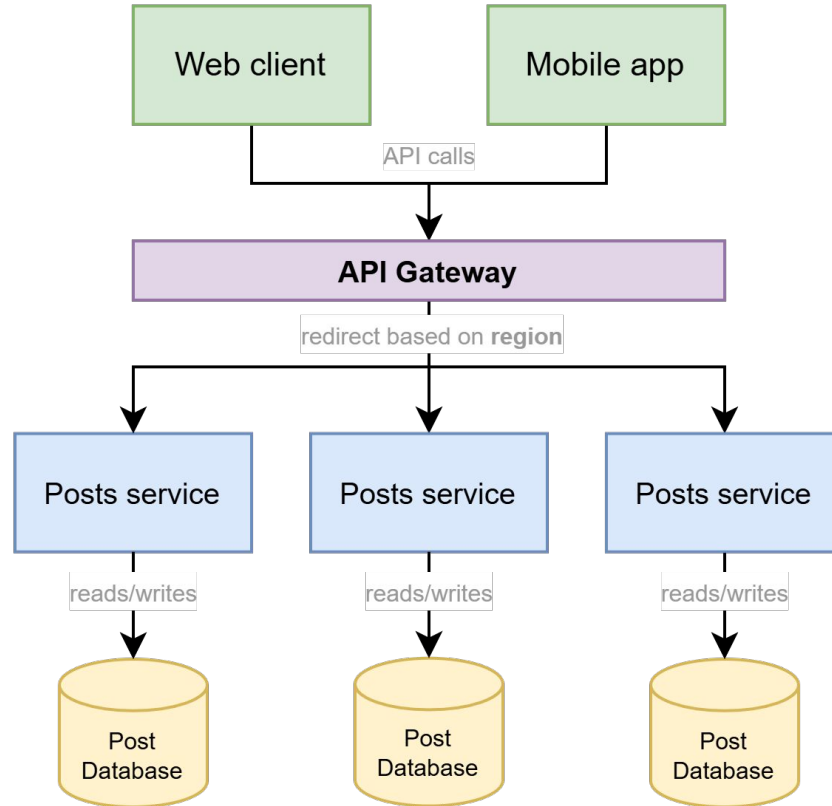
My Social Media



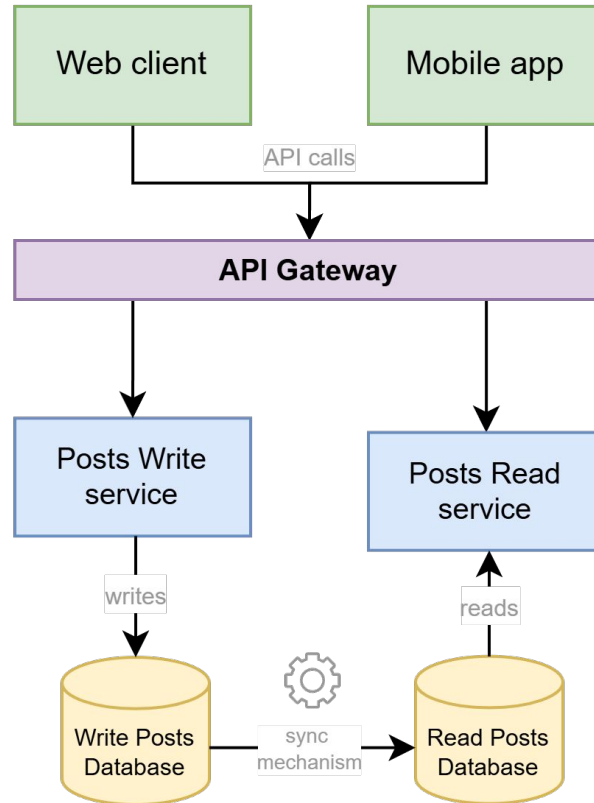
My Social Media



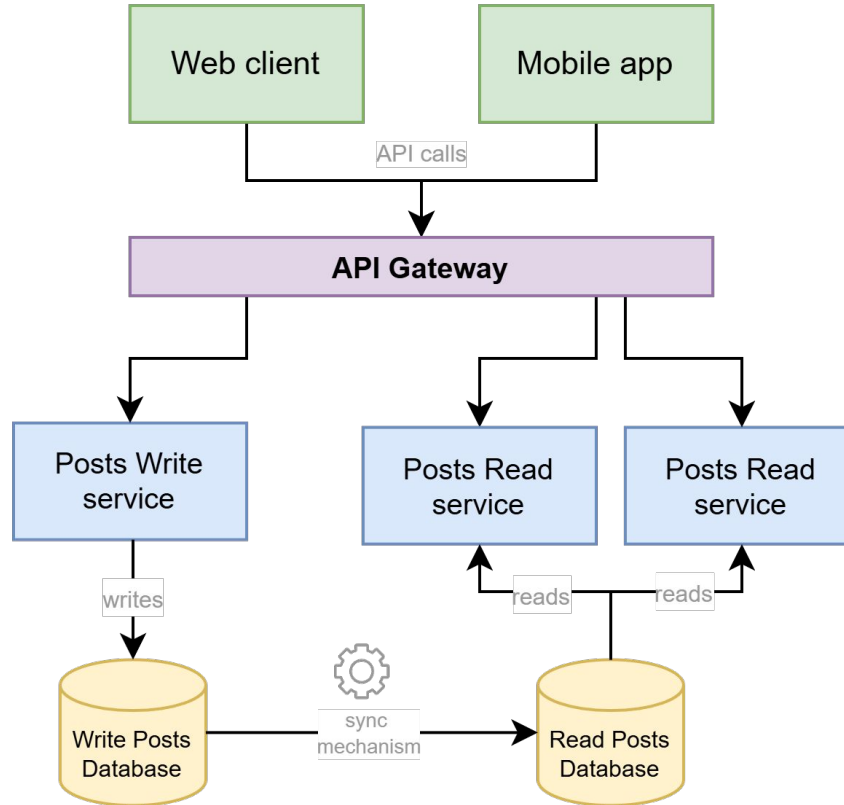
My Social Media



My Social Media



My Social Media



Mecanisme de sincronizarea a datelor

CQRS fara event sourcing

Abordare: SQL triggers

- Se folosesc triggere SQL pentru a înregistra modificările din baza de date *Write Posts Database* într-o tabelă secundară de evenimente.
- Evenimentele din această tabelă sunt **procesate periodic** (polling) pentru a sincroniza datele cu *Read Posts Database*.

```
CREATE OR REPLACE FUNCTION add_post_event_to_queue()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO EventQueue (AggregateId, EventType, EventData)
    VALUES (
        NEW.PostId,
        TG_OP || 'Post', -- TG_OP este comanda SQL (INSERT, UPDATE)
        to_jsonb(NEW)
    );
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

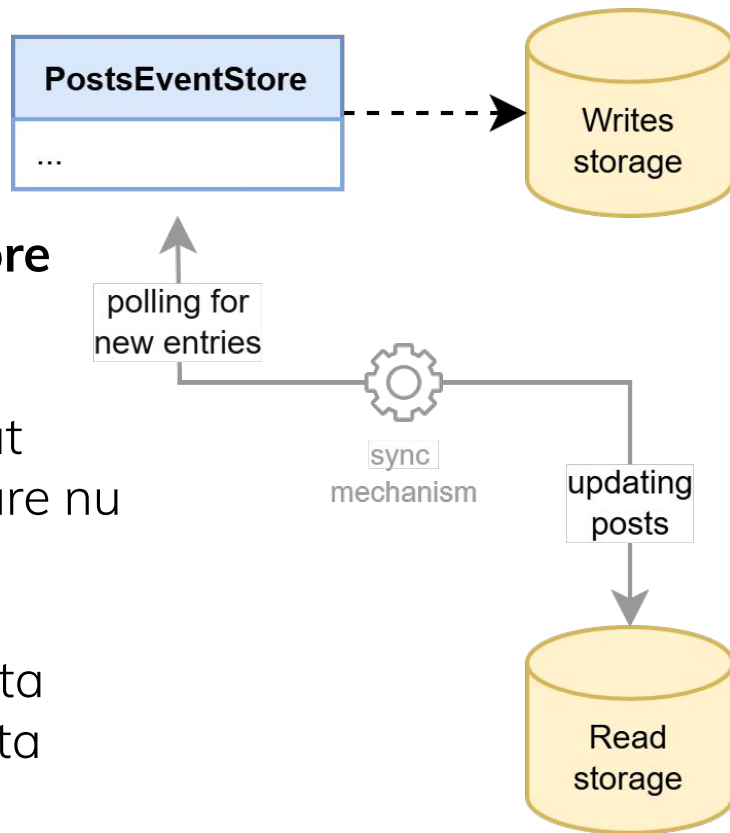
CREATE TRIGGER posts_event_trigger
AFTER INSERT OR UPDATE ON Posts
FOR EACH ROW
EXECUTE FUNCTION add_post_event_to_queue();
```

Mecanisme de sincronizarea a datelor

CQRS cu event sourcing

Abordare: Pooling Event Store

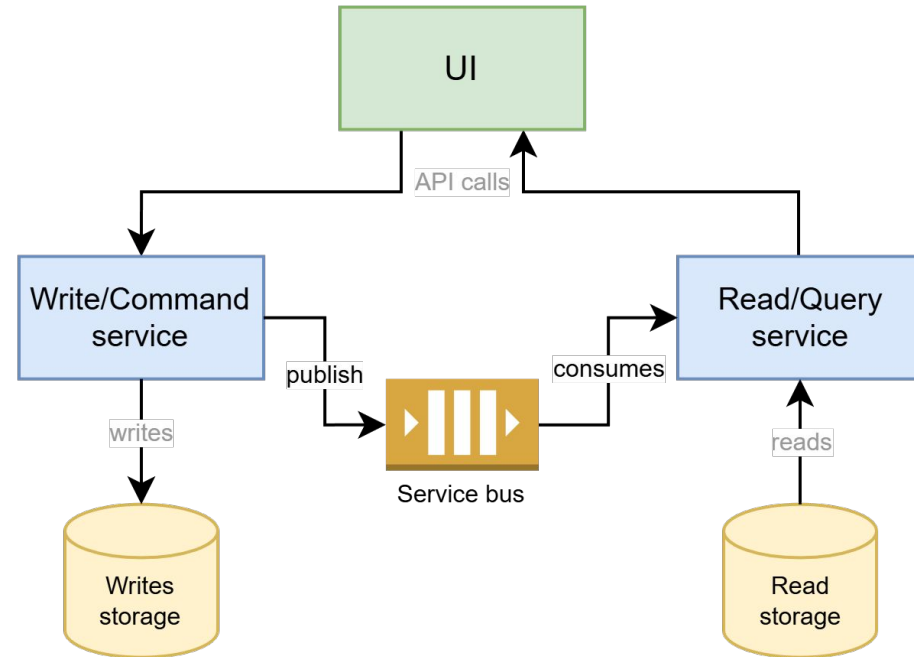
- Evenimentele sunt salvate în **Event Store** după procesarea unei comenzi.
- Un job de fundal sau un serviciu dedicat **preia evenimentele din Event Store** care nu au fost încă procesate și le aplică.
- După procesarea unui eveniment, acesta este **marcat** ca "procesat" pentru a evita dublarea.



Mecanisme de sincronizarea a datelor

Abordare: Trimiterea de evenimente

- După fiecare operație de scriere, **un evenimentul este publicat** într-un sistem de mesagerie (ex.: RabbitMQ, Kafka, Azure Service Bus).
- **Posts read service** consumă evenimentul și actualizează baza de date de citire.



Demo

