



Event Sourcing

Historia pewnej encji

Jakub Saleniuk



Historia pewnej encji


$$2+2=?$$

© 2004 Ted Goff

Records Request



**"You want to review the
association's records?"**



“Event Sourcing zapewnia, że wszystkie zmiany stanu aplikacji zostają zapisane jako sekwencja zdarzeń. Nie tylko możemy pobrać listę tych zdarzeń, lecz także wykorzystać je do odtworzenia stanu z przeszłości...”

- Martin Fowler





prooph



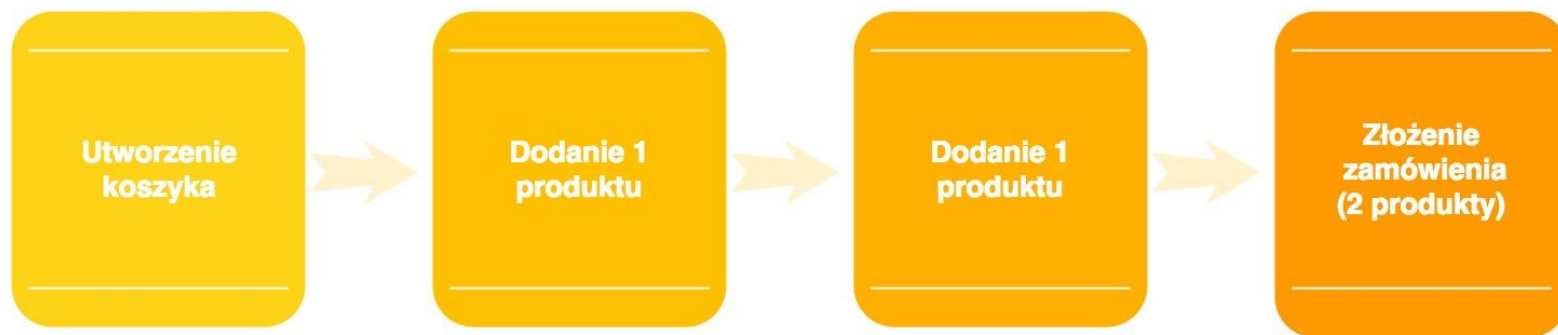
■ AD-Automatic Deposit ■ AP-Automatic Payment ■ ATM-Teller Machine ■ DC-Debit Card ■ T-Tax Deductible

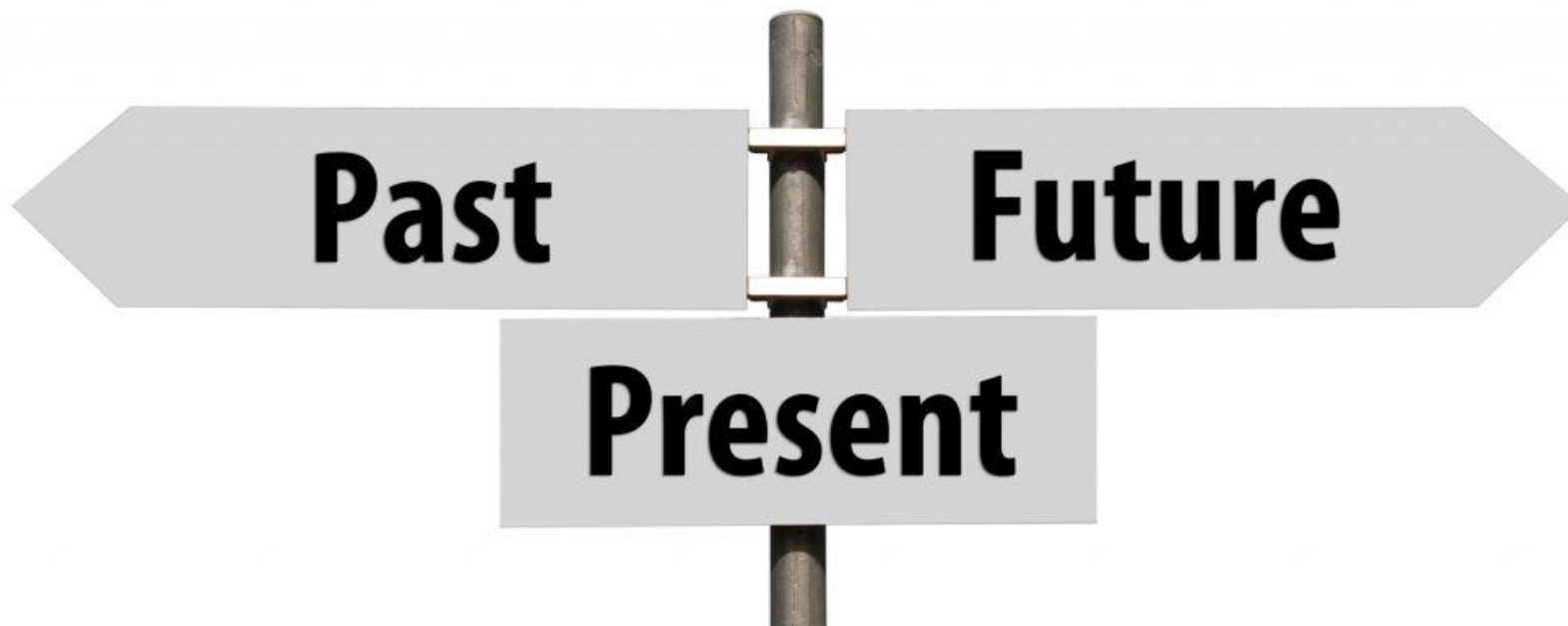
NUMBER OR CODE	DATE	TRANSACTION DESCRIPTION	PAYMENT AMOUNT	✓	FEE	DEPOSIT AMOUNT
001	5/1	Mortgage	\$123612			\$
002	5/8	Supermarket (food)	8751			
003	5/11	Electricity	12785			
004	5/16	Real Estate Tax	35917			



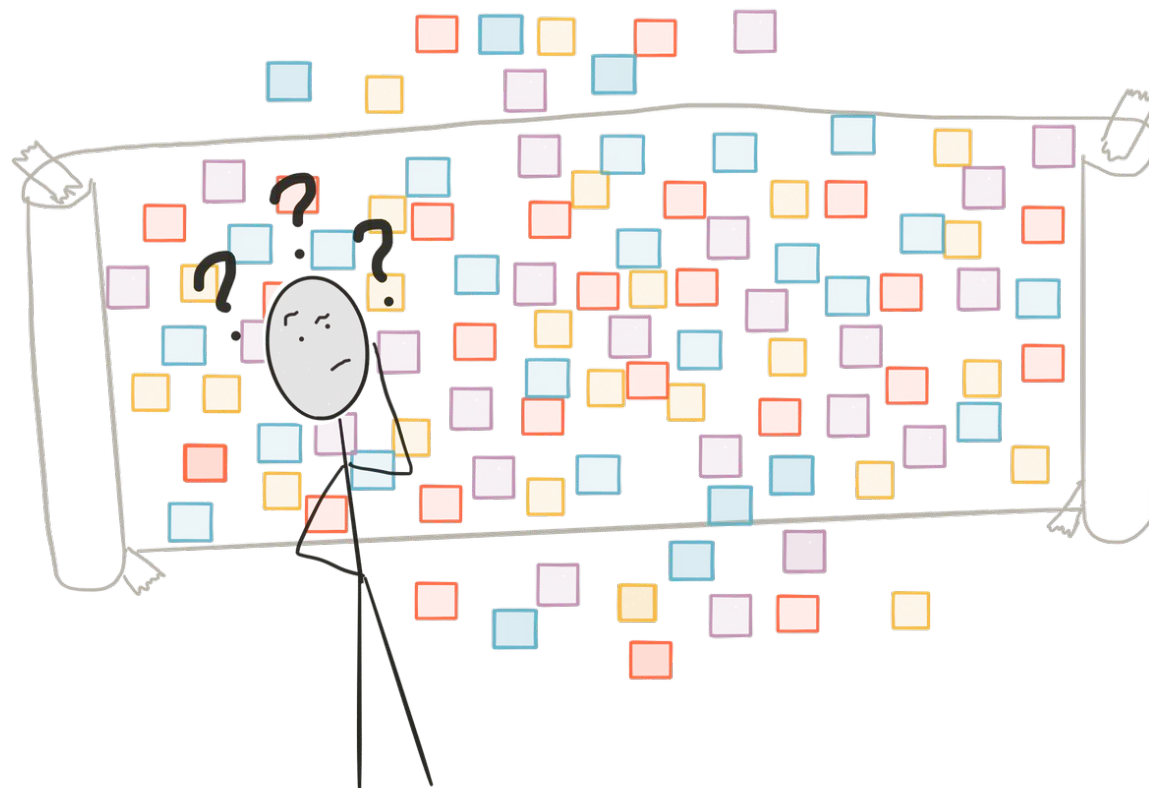
Event Sourcing **nie traci** informacji!







Event Storming



Event Sourcing - projekcja encji

Konto bankowe

id	użytkownik	saldo	status	data_dodania
1	1	4000	1	06-10-2011

Eventy

id	nazwa	aggregateId	event	data_dodania	BankAccount	
1	BankAccountWasRegistered	1	{ "użytkownik": "1" }	06-10-2011	id	1
2	BankAccountWasLoaded	1	{ "kwota": "10000" }	07-10-2011	użytkownik	1
3	BankAccountWasUnloaded	1	{ "kwota": "6000" }	08-10-2011	saldo	40000
4	BankAccountWasClosed	1	{ }	12-10-2011	status	2

BankAccountProjector

```
class BankAccountProjector implements ProjectorInterface
{
    // fields and constructor

    public function applyEvents(EventAggregateInterface $eventAggregate): AggregateInterface
    {
        foreach ($eventAggregate->getEvents() as $event) {
            $this->applyEvent($event);
        }

        return $this->bankAccount;
    }

    // other methods
}
```

BankAccountProjector

```
class BankAccountProjector implements ProjectorInterface
{
    // fields and constructor

    public function applyEvent(EventInterface $event): AggregateInterface
    {
        $methodName = 'apply' . $event->getEventName();
        $this->$methodName($event);

        return $this->bankAccount;
    }

    // other methods
}
```

BankAccountProjector

```
class BankAccountProjector implements ProjectorInterface
{
    // fields and constructor

    public function applyBankAccountWasRegistered(BankAccountWasRegistered $event)
    {
        $bankAccount = new BankAccount(
            $event->getAggregateId(),
            $event->getUserId(),
            $event->getName(),
            0,
            1
        );

        $this->bankAccount = $bankAccount;
    }

    // other methods
}
```


BankAccountProjector

```
class BankAccountProjector implements ProjectorInterface
{
    // fields and constructor

    public function applyBankAccountWasLoaded(BankAccountWasLoaded $event)
    {
        $this->bankAccount->loadBankAccount(
            $event->getAmount()
        );
    }

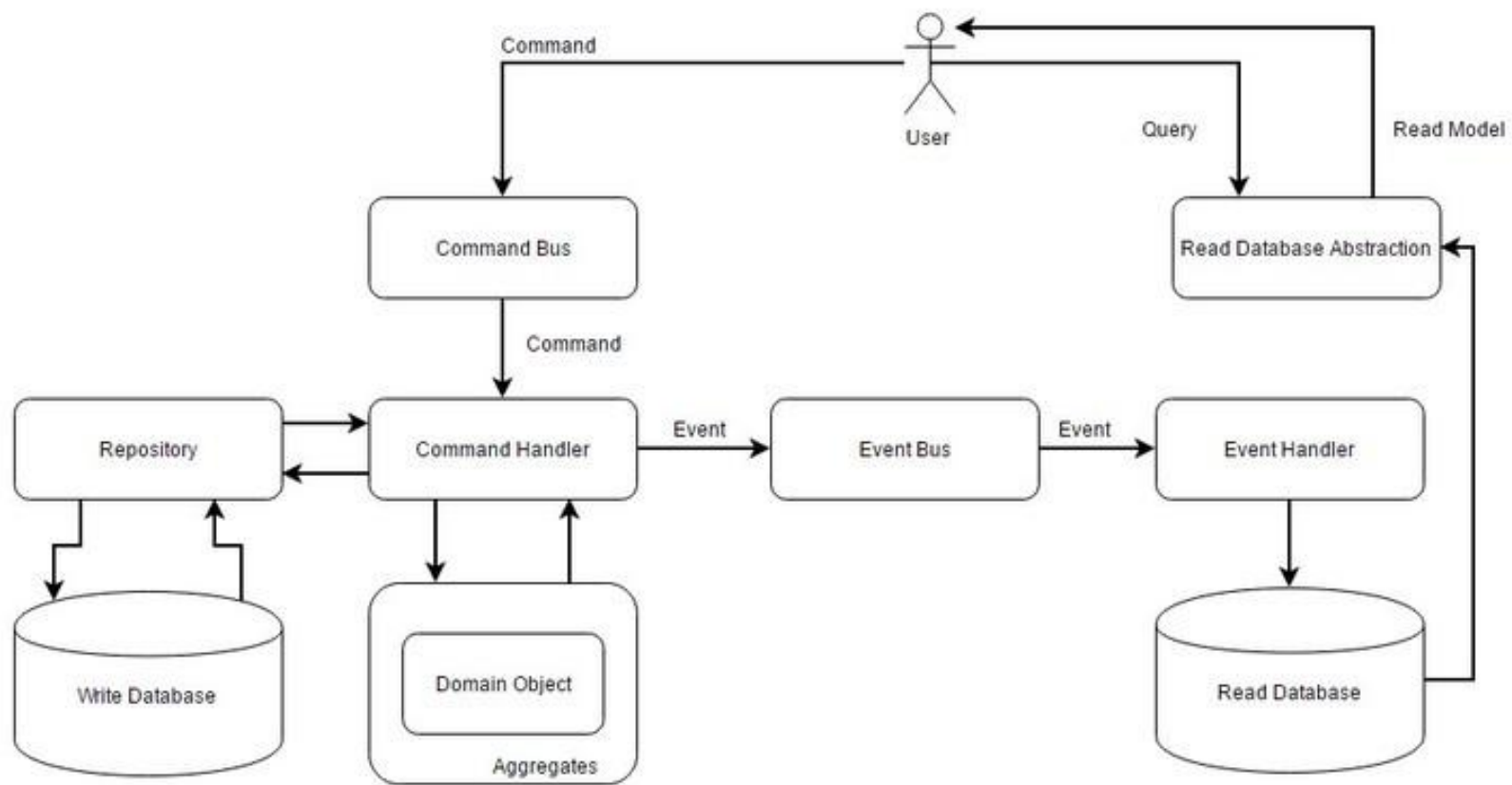
    // other methods
}
```

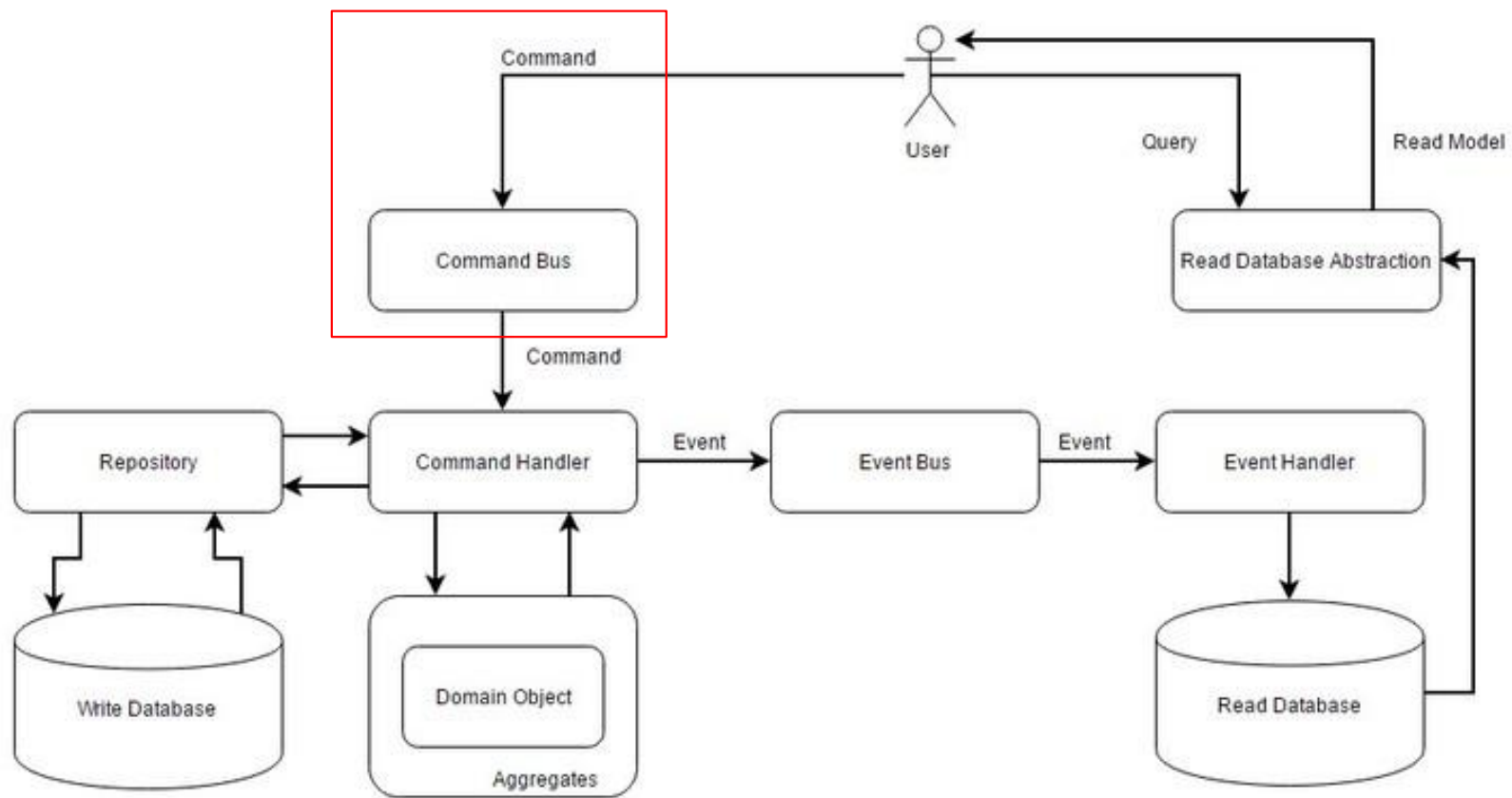
BankAccount

```
class BankAccount
{
    // fields and constructor

    public function loadBankAccount(int $amount)
    {
        $this->balance += $amount;
    }

    // other methods
}
```





RegisterBankAccountCommand

```
class RegisterBankAccountCommand
{
    private $userId;

    private $name;

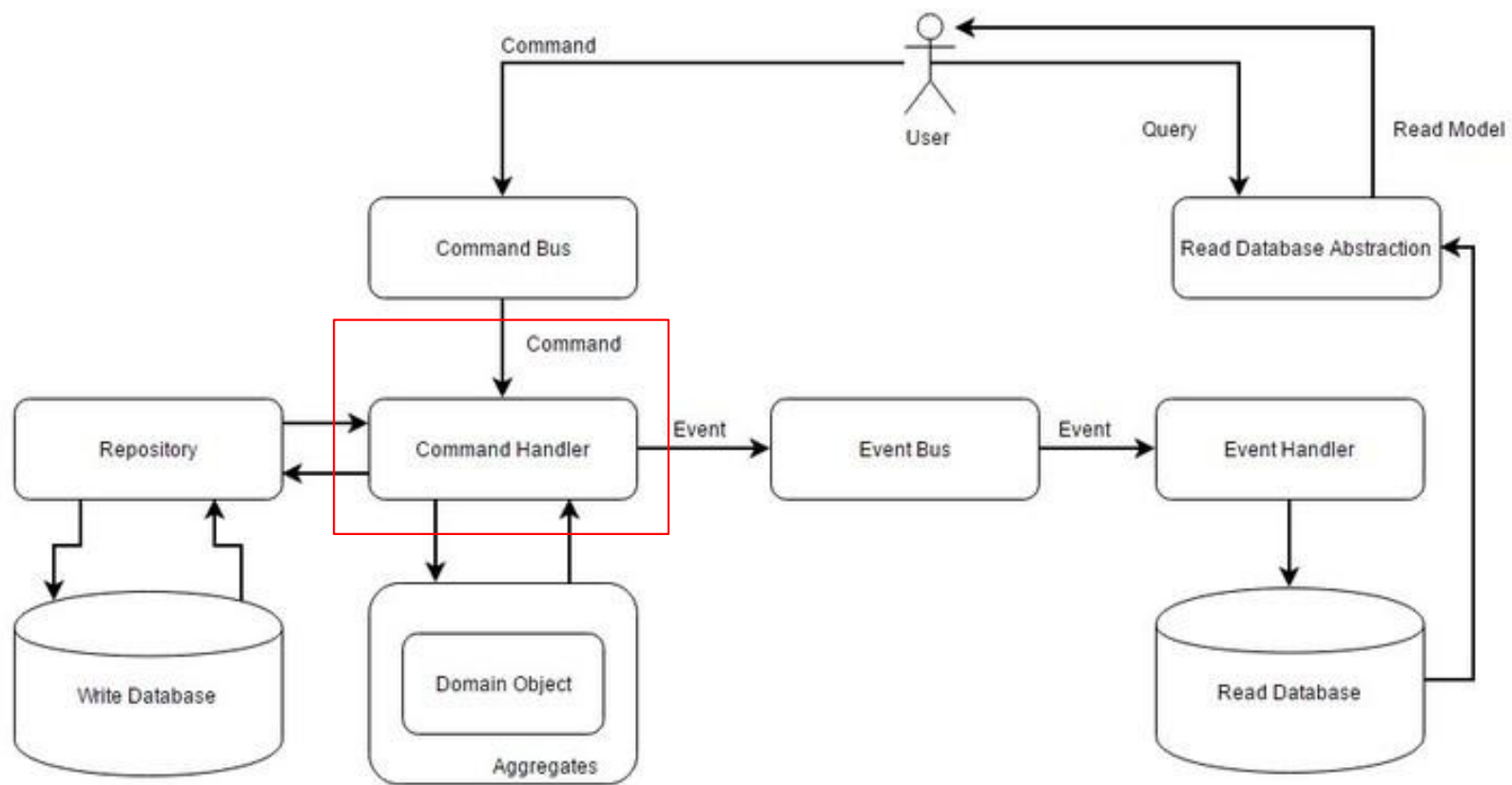
    public function __construct($userId, $name)
    {
        $this->userId = $userId;
        $this->name = $name;
    }

    // getters and setters
}
```

BankAccountController

```
class BankAccountController extends Controller
{
    public function register(Request $request, RegisterBankAccount $registerBankAccount)
    {
        $command = new RegisterBankAccountCommand(
            (int)$request->get('user_id'),
            $request->get('name')
        );
        $registerBankAccount->execute($command);
    }

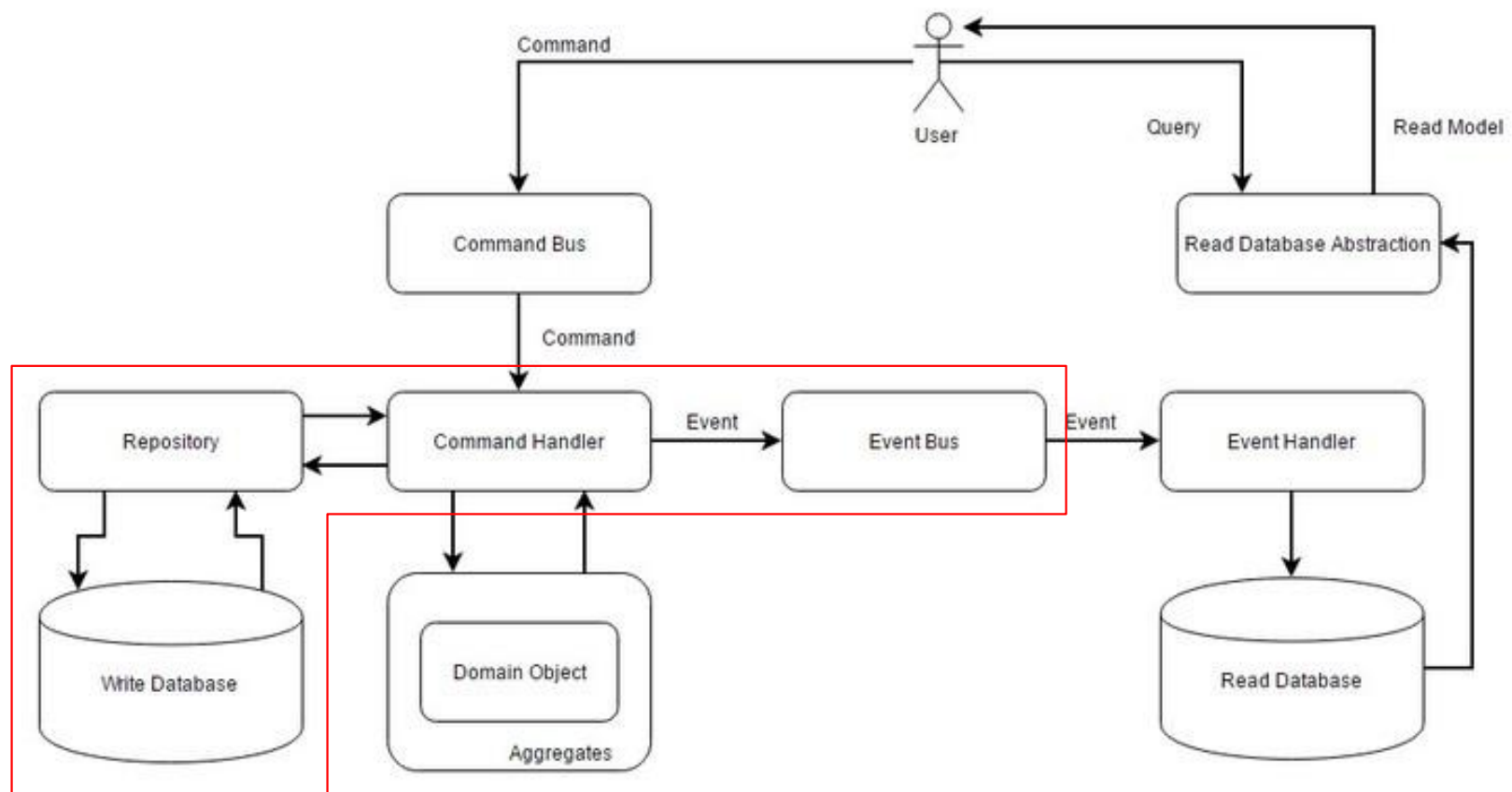
    // other methods
}
```



RegisterBankAccount

```
class RegisterBankAccount implements RegisterBankAccountInterface
{
    // fields and constructor

    public function execute(RegisterBankAccountCommand $command)
    {
        $bankAccount = $this->bankAccountFactory->create(
            $command->getUserId(),
            $command->getName()
        );
        $this->bankAccountRepository->register($bankAccount);
    }
}
```



BankAccountRepository

```
class BankAccountRepository implements BankAccountRepositoryInterface
{
    // fields and constructor

    public function register(BankAccount $bankAccount)
    {
        $bankAccountWasRegistered = $this->bankAccountWasRegisteredFactory->create(
            $bankAccount->getUserId(),
            $bankAccount->getName()
        );
        $this->dispatchEventService->dispatch($bankAccountWasRegistered);

        return $bankAccountWasRegistered->getId();
    }

    // other methods
}
```

DispatchEventService

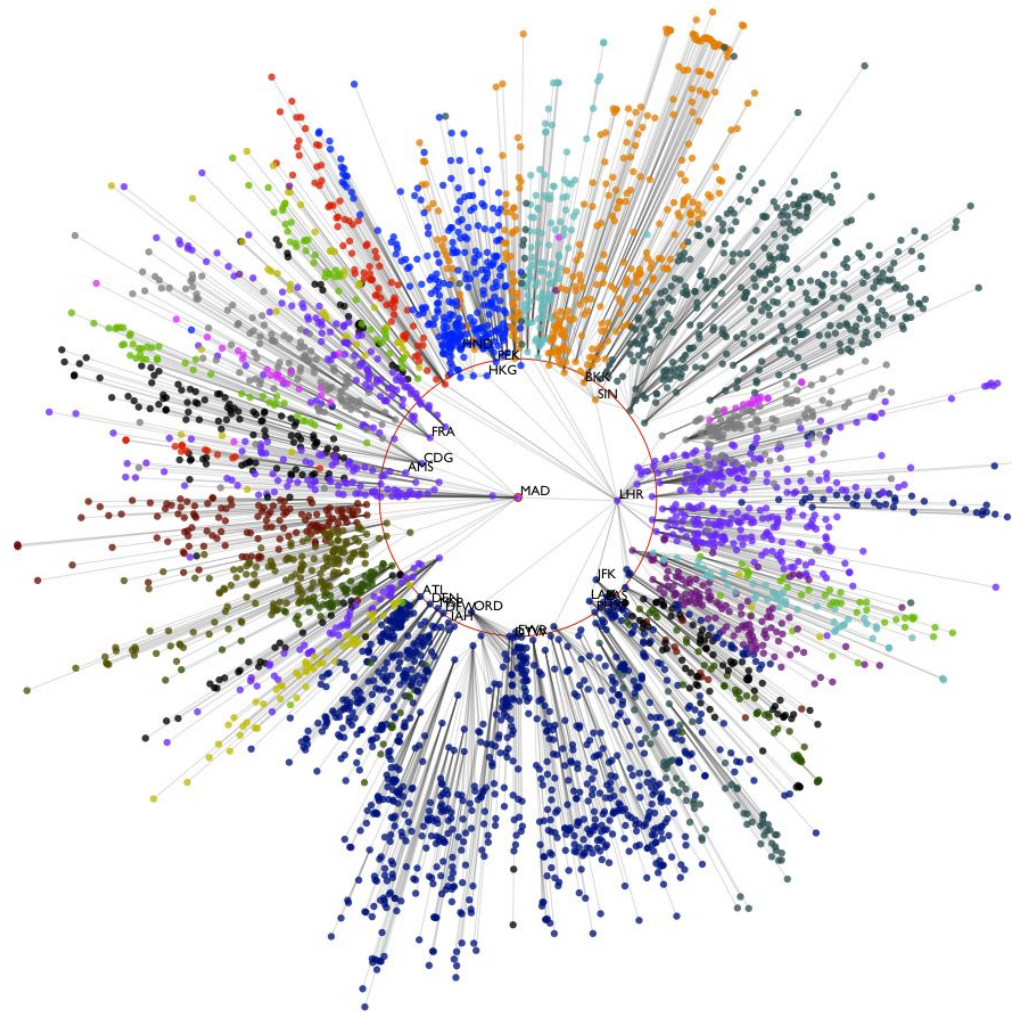
```
class DispatchEventService implements DispatchEventServiceInterface
{
    // fields and constructor

    public function dispatch(EventInterface $event)
    {
        try {
            DB::beginTransaction();

            $eventStorage = $this->eventStorageFactory->create($event);
            $id = $this->eventStorageRepository->save($eventStorage);

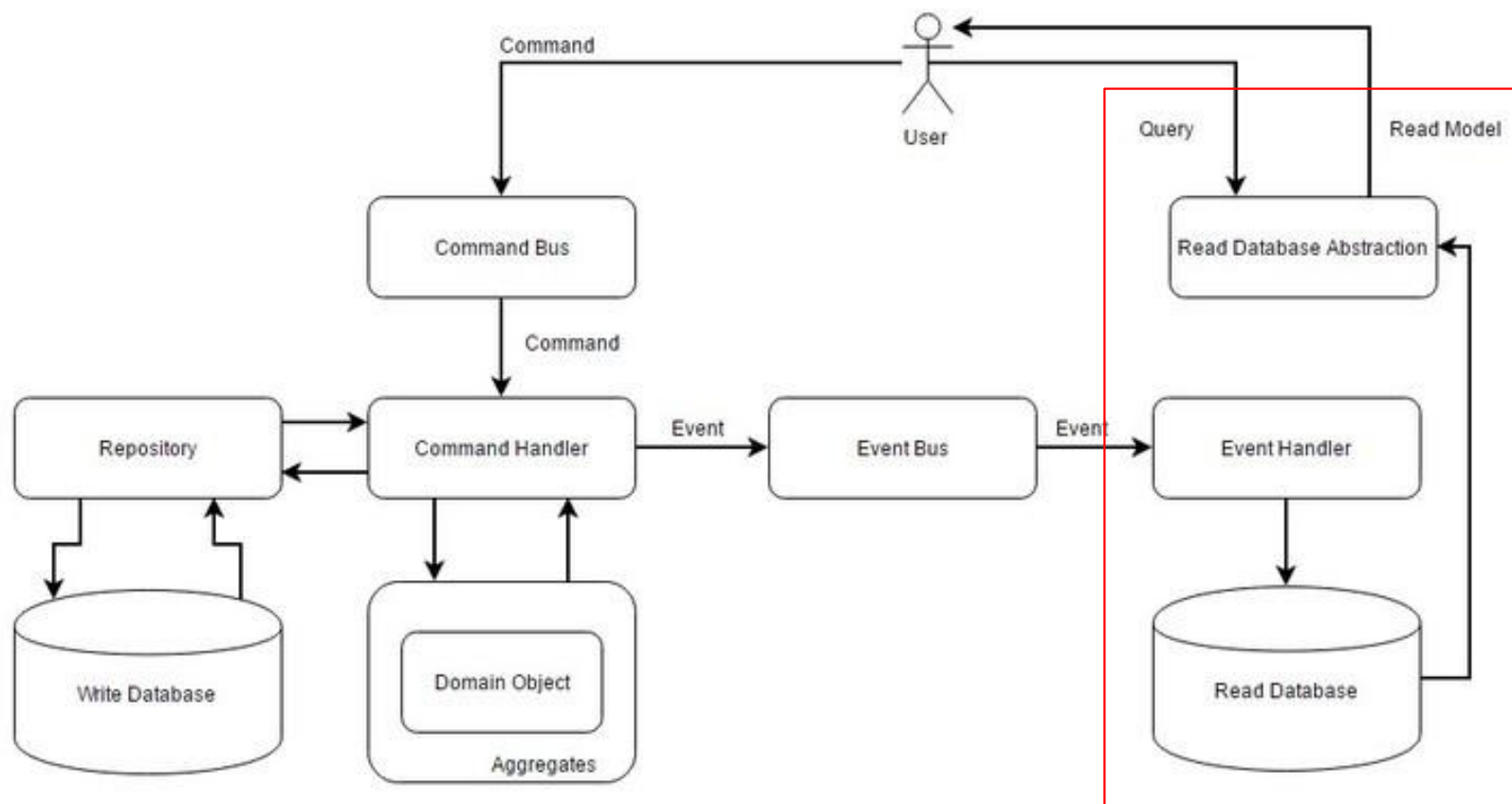
            $event->setId($id);
            $this->eventBus->dispatch($event);

            DB::commit();
        } catch (\Exception $exception) {
            DB::rollBack();
            throw new EventNotDispatchedException();
        }
    }
}
```











elasticsearch





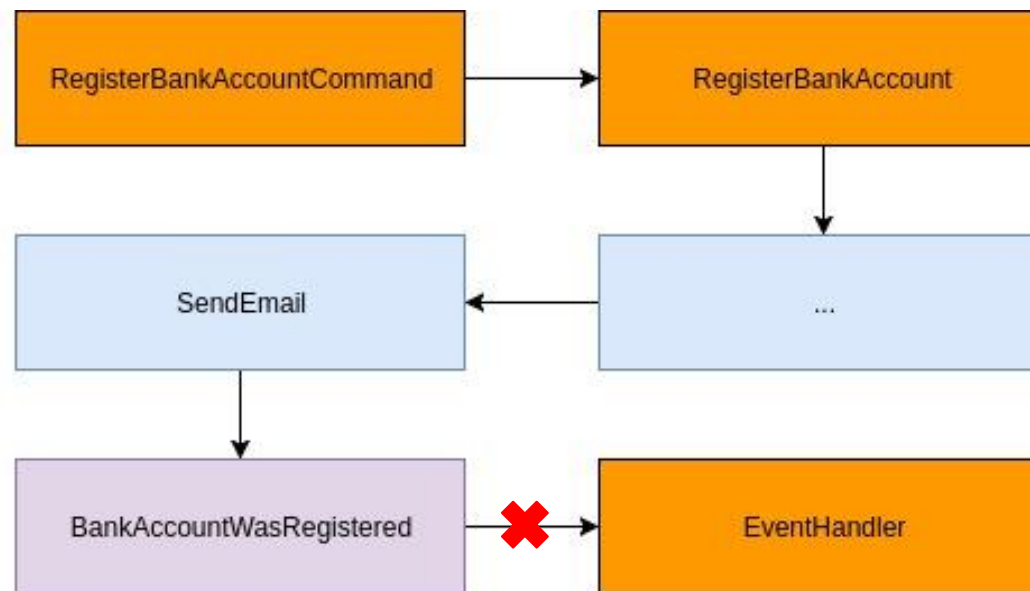




```
{  
  "uzytkownik": "1",  
  "imie": "Jan",  
  "nazwisko": "Kowalski"  
}
```

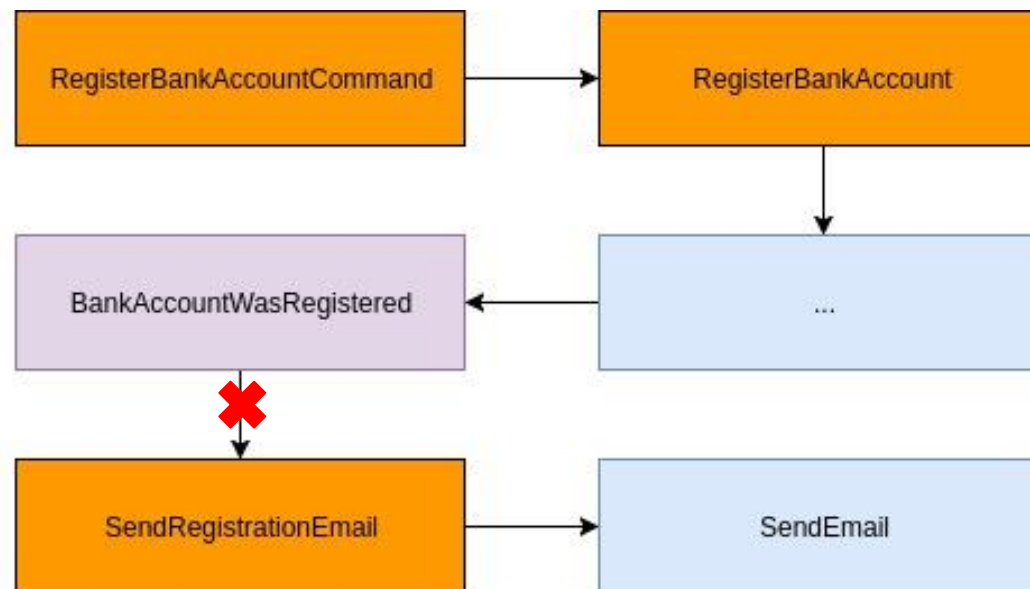
```
{  
  "uzytkownik": "1",  
  "imie": "Jan",  
  "nazwisko": "Kowalski",  
  "email": "jan@kowalski.pl"  
}
```

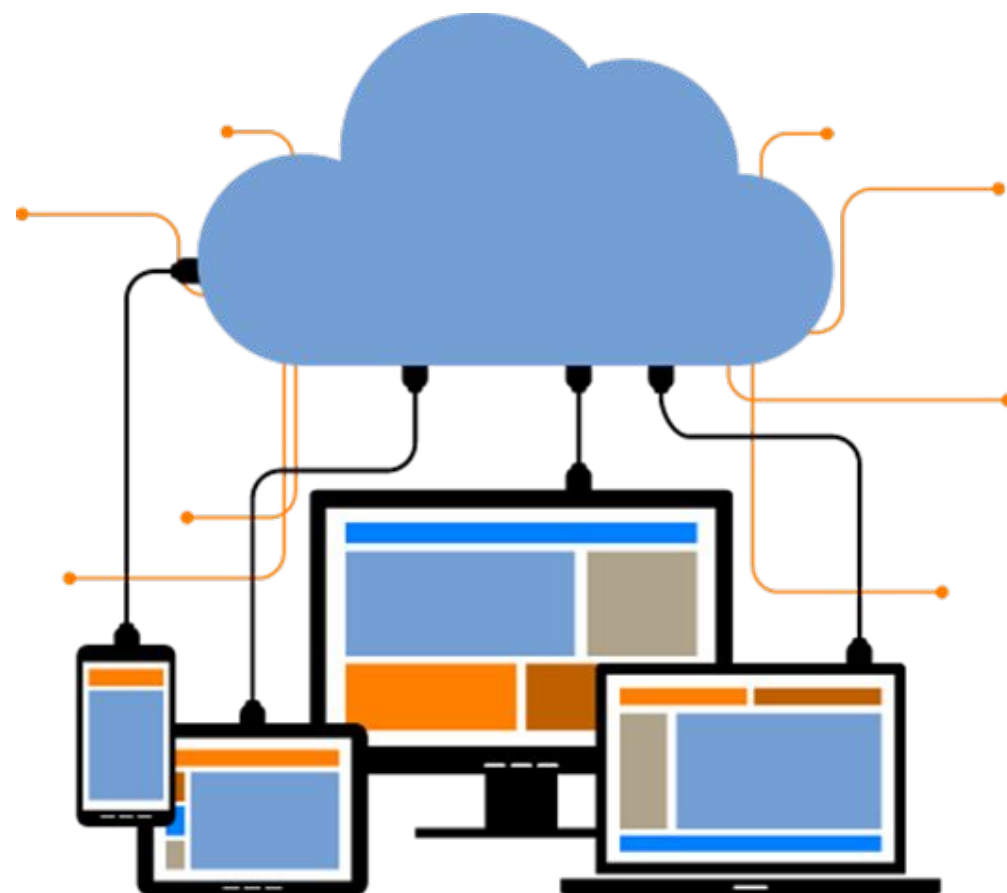


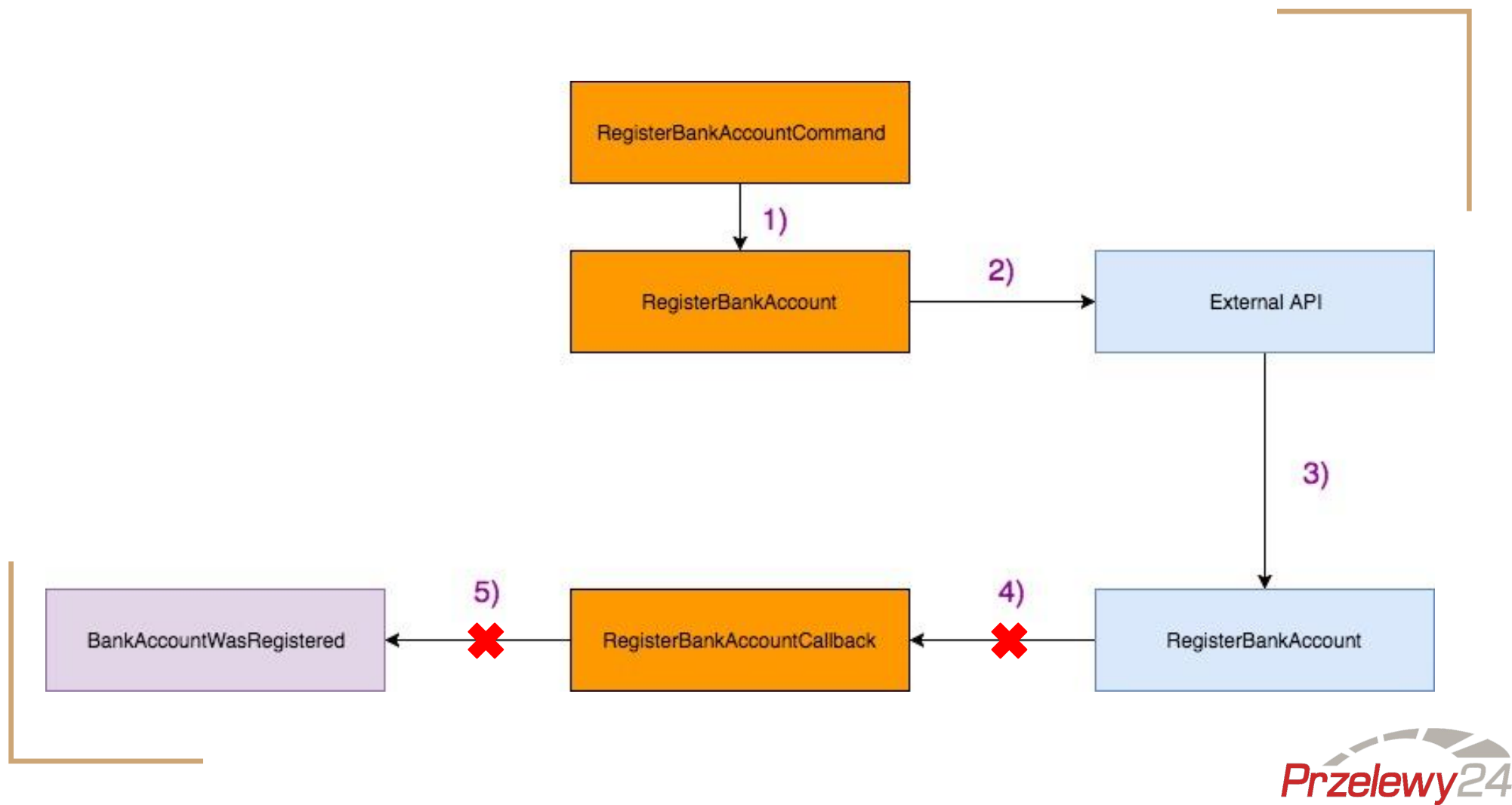



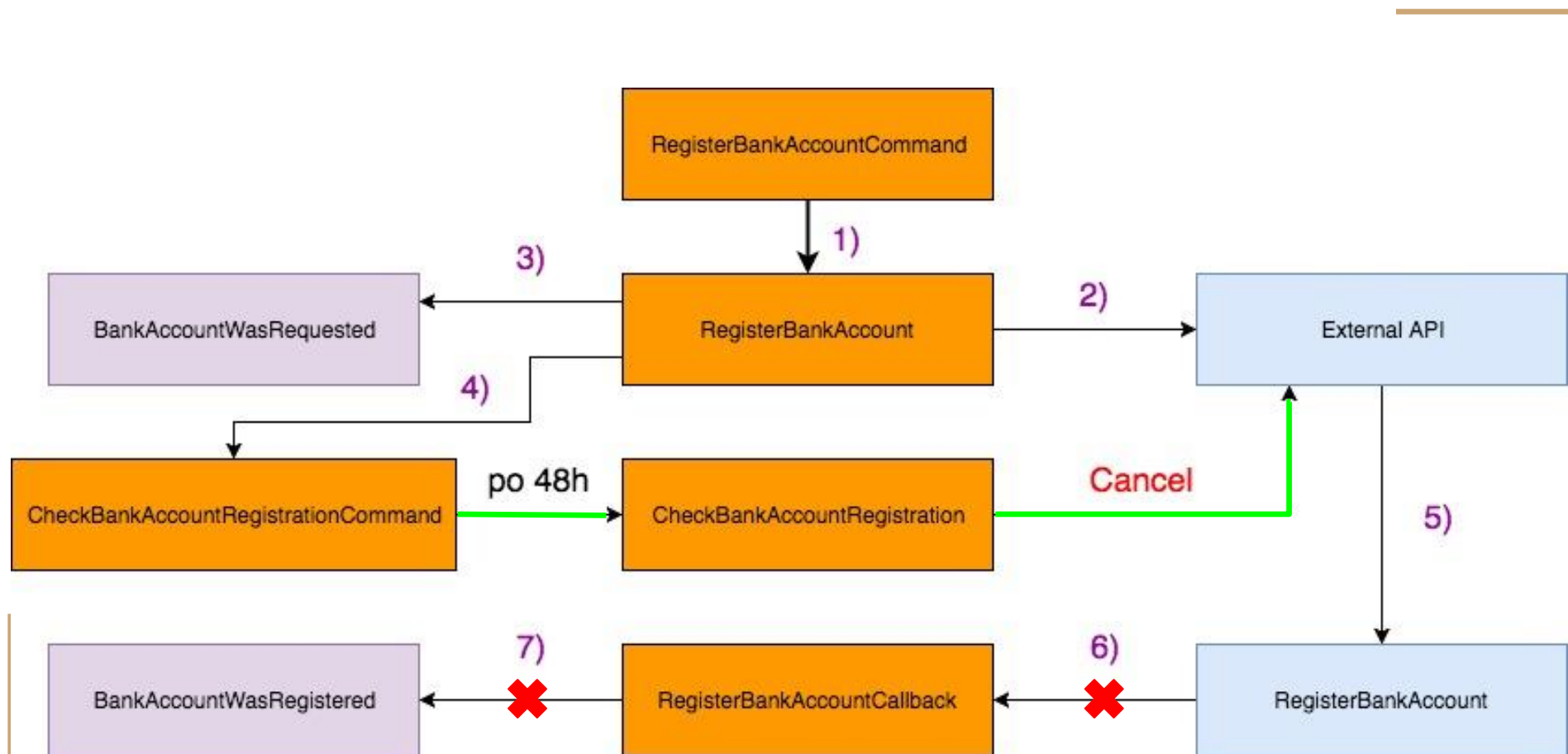












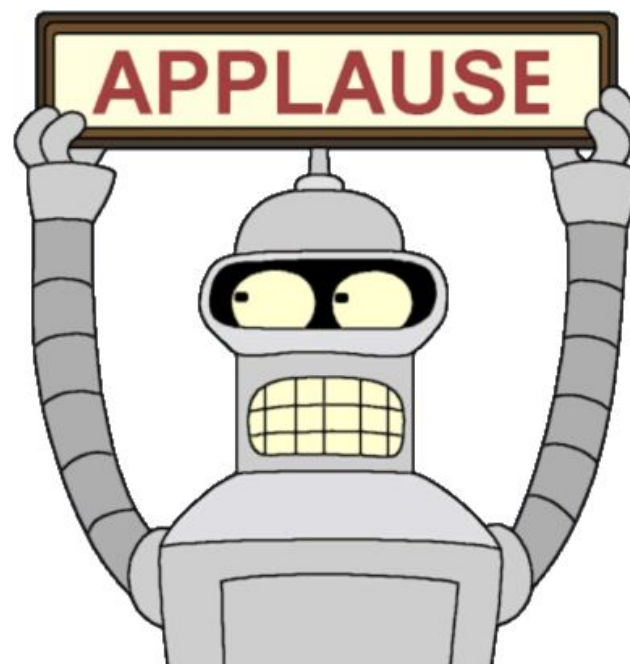


<https://jsaleniuk.wordpress.com/>









Koniec