The Messenger

Symfony Component

Do CQRS they say, it's cool!

Žilvinas Kuusas / VilniusPHP 0x5d

About me

Žilvinas Kuusas

Software engineer since 2004

Currently DevOps @ Dokobit

You may know me from: VilniusPHP.lt, NoTrollsAllowed.com, Symfony.lt, CleanPHP.lt ...

linkedin.com/in/kuusas

Electronic document signing platform

Biggest in Baltic countries and Iceland.

Expanding in Norway, Sweden, Poland…

Currently have 15+ open job positions.

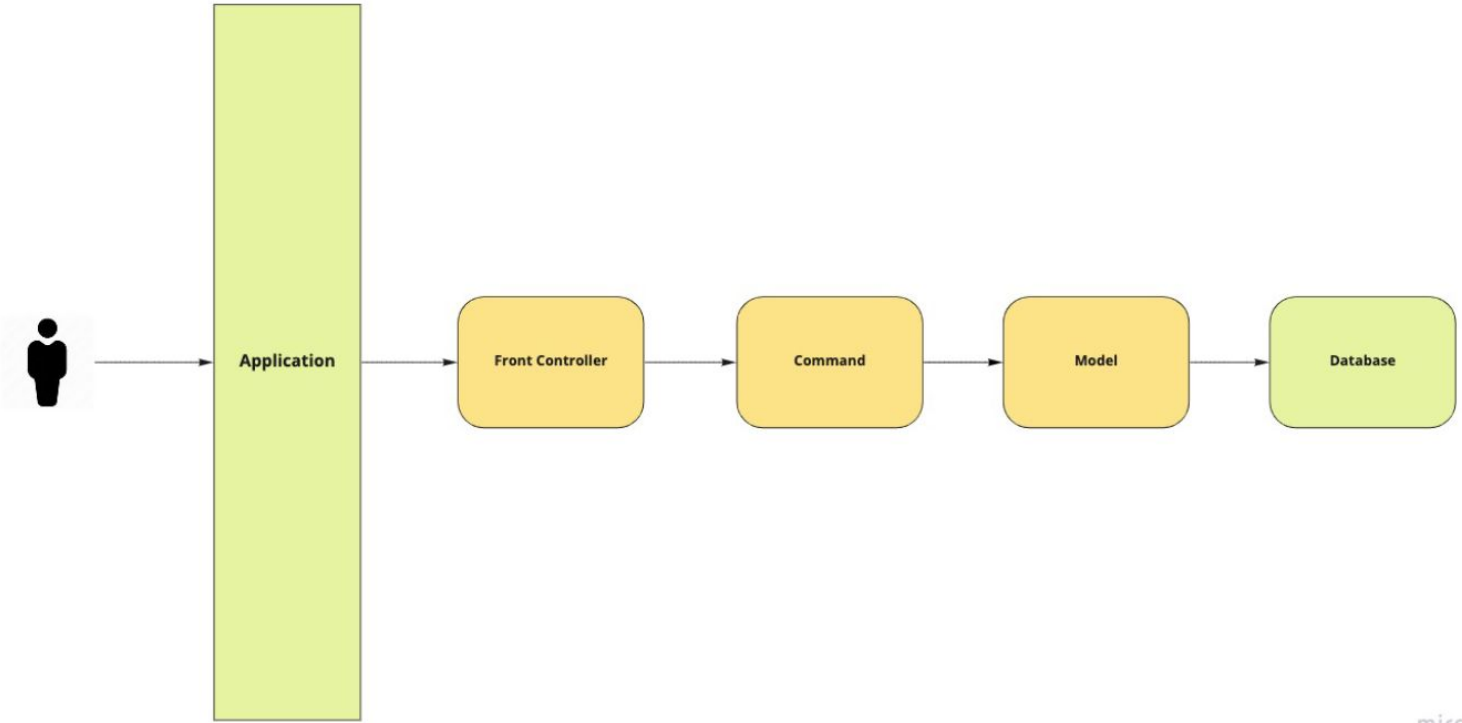
Today’s talk overview

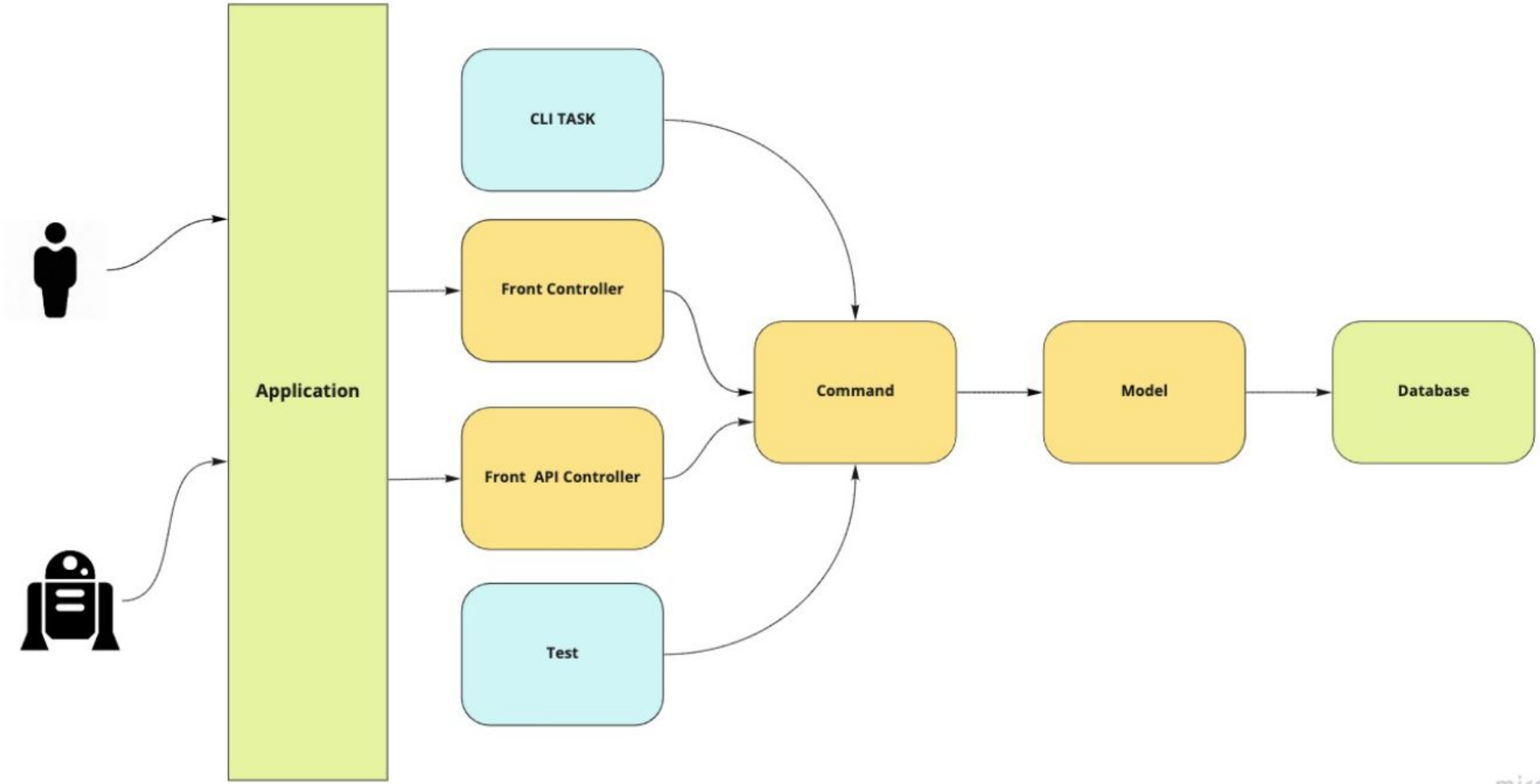
Some Design Patterns

How to start using The Messenger Component

Do crazy things with your Business Context

Command Bus Pattern




CLI TASK
Front Controller
Front API Controller
Test
Application
Command
Model
Database

Command Bus Pattern

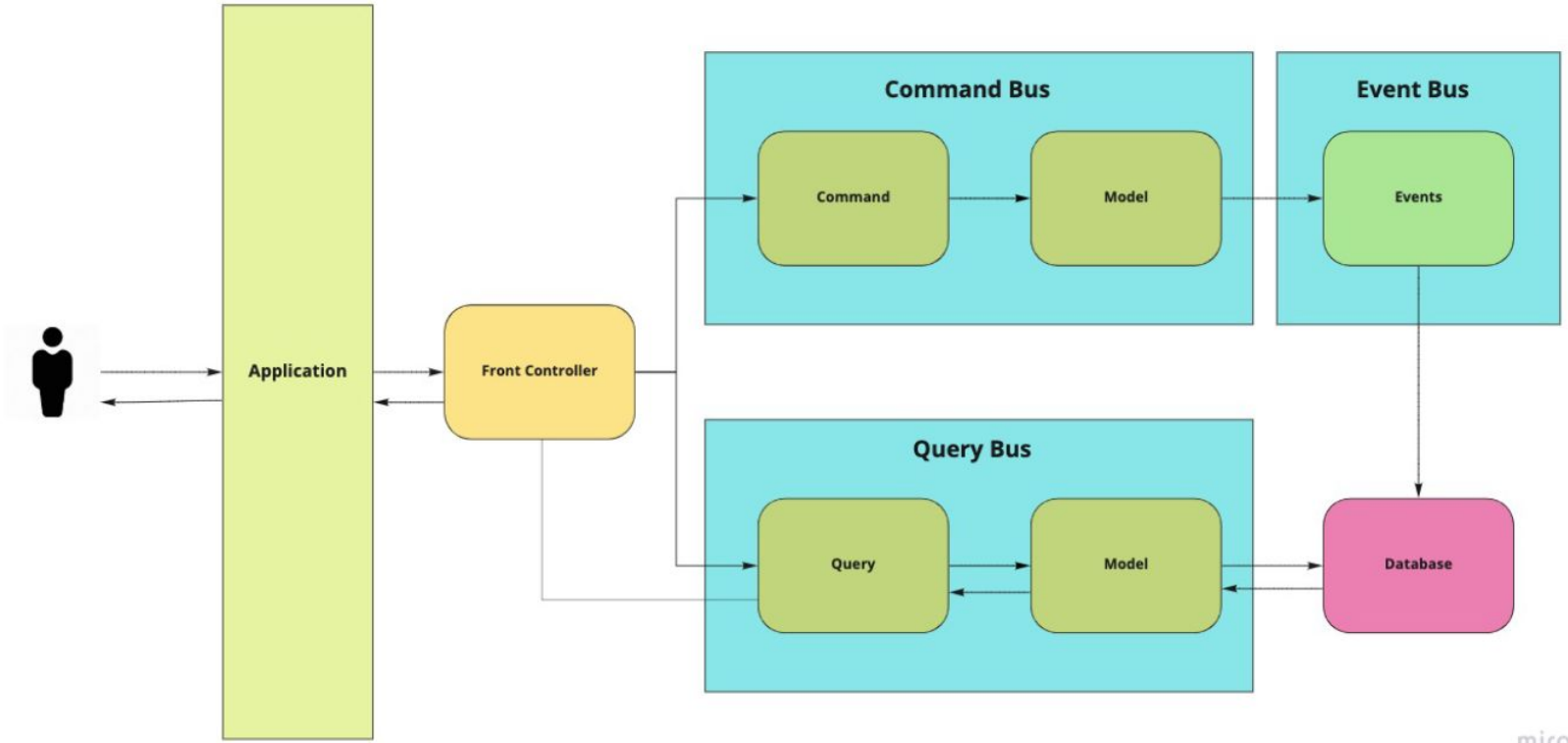
Why we should use it?

- Decoupled
- Testable
- Extendable

Downsides

- Complexity

CQRS


Application
Front Controller
Command Bus
Command
Model
Event Bus
Events
Query Bus
Query
Model
Database

How to CQRS CRUD application?

Don’t.

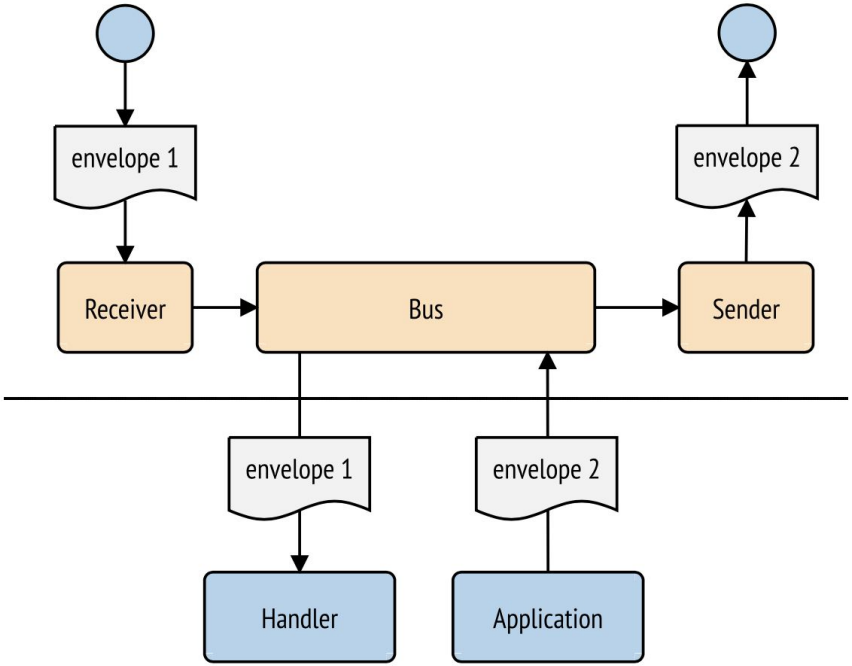
The Messenger

The Messenger

Simple yet very powerful and flexible message bus manager.

“The Messenger component helps applications send and receive messages to/from other applications or via message queues.”

Concepts


envelope 1
envelope 2
Receiver
Bus
Sender
envelope 1
envelope 2
Handler
Application

Command

What To Do?

The message you want to pass to the bus.

Message will be consumed and handled the way command bus is configured.

Message Bus

Where To Execute?

An ordered set of middlewares:

1. messenger.middleware.add_bus_name_stamp_middleware
2. messenger.middleware.dispatch_after_current_bus
3. messenger.middleware.failed_message_processing_middleware
4. … custom middlewares...
5. messenger.middleware.send_message
6. messenger.middleware.handle_message

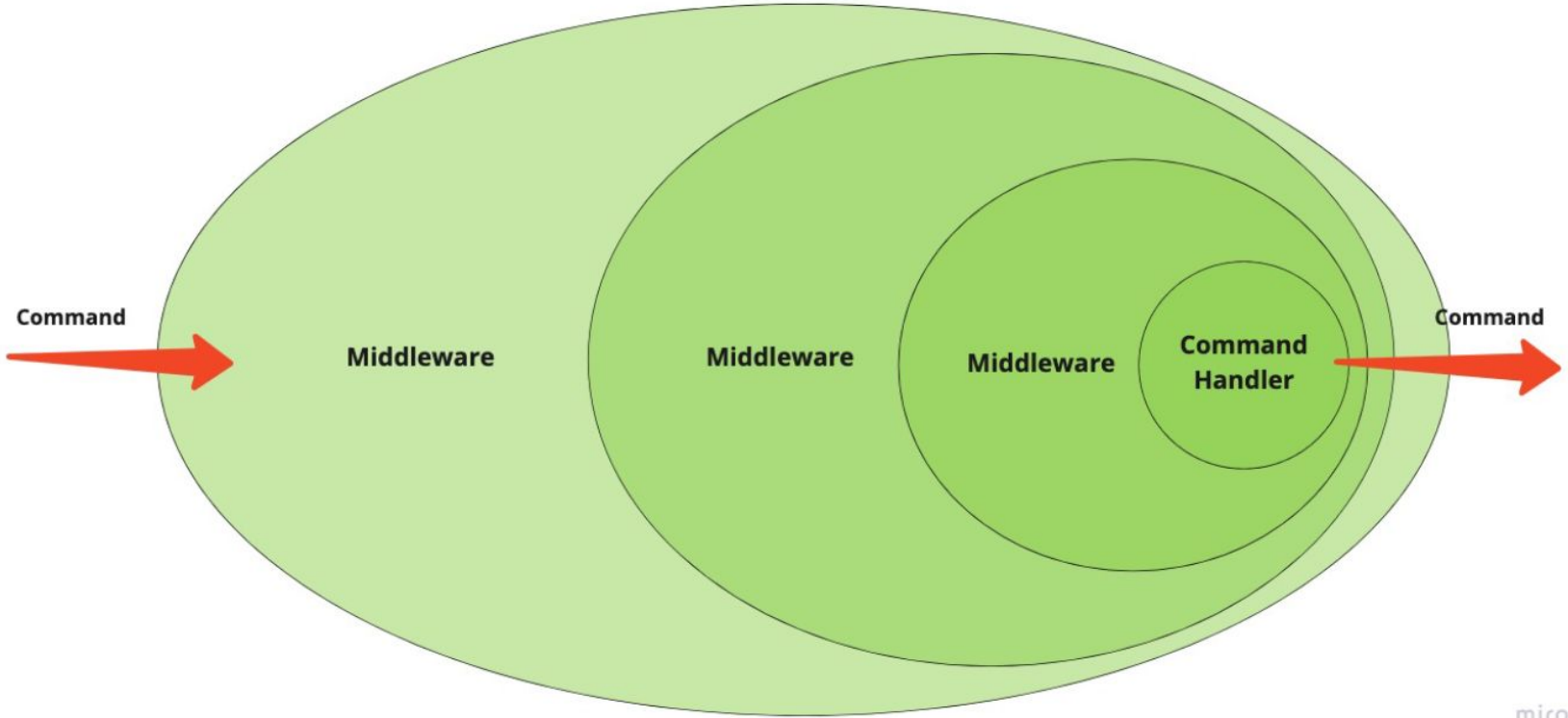
Message Bus

```php
$handler = new MyMessageHandler();

$bus = new MessageBus([
    new HandleMessageMiddleware(new HandlersLocator([
        MyMessage::class => [$handler],
    ])),
]);

$bus->dispatch(new MyMessage(/* ... */));
```

Message Bus



Middleware

Software in-the-middle.

Can access the message and its wrapper.

No business logic.

Affects entire bus flow.

Middleware

Validate messages

Drop messages

Log messages

Middleware are called both when a message is originally dispatched and again later when a message is received from a transport,

Handler

What To Do?

The command executor, the business model…

Each **Command** should have **One Handler**.

Events

Command should execute single task

All the secondary tasks should be decoupled using Event

Start Using It

Step 1: install

```
$ composer require symfony/messenger
```

Step 2: create command

```php
namespace App\Message;

class SendEmail
{
    public function __construct(string $email)
    {
        $this->email = $email;
    }

    public function getEmail()
    {
        return $this->email;
    }
}
```

Step 3: create handler

```php
namespace App\MessageHandler;

use App\Message\SendEmail;
use Symfony\Component\Messenger\Handler\MessageHandlerInterface;

class SendEmailHandler implements MessageHandlerInterface
{
    public function __invoke(SendEmail $message)
    {
        // ... do some work - like sending an email
    }
}
```

Step 4: dispatch command

```php
public function __construct(MessageBusInterface $bus)
{
    $this->bus = $bus;
}

protected function execute(InputInterface $input, OutputInterface $output): int
{
    $io = new SymfonyStyle($input, $output);
    $email = $input->getArgument('email');

    $io->note(sprintf('Sending email to: %s', $email));

    $this->bus->dispatch(new SendEmail($email));

    $io->success('Email queued.');

    return 0;
}
```

Step 5: test

```
$ bin/console app:send-email zilvinas@kuusas.lt

 ! [NOTE] Sending email to: zilvinas@kuusas.lt

object(App\Message\SendEmail)#60 (1) {
  ["email"]=>
  string(18) "zilvinas@kuusas.lt"
}

 [OK] Email queued.
```

Step 6: async

config/packages/messenger.yml

```yaml
framework:
    messenger:
        transports:
            async: 'amqp://user:pwd@rabbitmq:5672/%2f/messages'

        routing:
            'App\Message\SendEmail': async
```

Step 7: test

```
$ bin/console app:send-email zilvinas@kuusas.lt

 ! [NOTE] Sending email to: zilvinas@kuusas.lt

 [OK] Email queued.
```

Step 8: consume

```
$ bin/console messenger:consume async -vv

 [OK] Consuming messages from transports "async".

 // The worker will automatically exit once it has received a stop signal via the messenger:stop-workers command.

 // Quit the worker with CONTROL-C.

[info] Received message App\Message\SendEmail

object(App\Message\SendEmail)#123 (1) {
  ["email"]=>
  string(18) "zilvinas@kuusas.lt"
}
[info] Message App\Message\SendEmail handled by App\MessageHandler\SendEmailHandler::__invoke

[info] App\Message\SendEmail was handled successfully (acknowledging to transport).
```

Step 9: multiple transports

```yaml
framework:
    messenger:
        transports:
            async: 'amqp://user:pwd@rabbitmq:5672/%2f/messages'
            sync: 'sync://'

        routing:
            'App\Message\SendEmail': async
            'App\Message\ShowNotification': sync
```

Step 10: getting result from bus

```php
$envelope = $this->bus->dispatch(new ShowNotification($userId));

$handledStamp = $envelope->last(HandledStamp::class);

$result = $handledStamp->getResult();
```

Step 11: multiple buses

```yaml
framework:
    messenger:
        default_bus: command.bus
        buses:
            command.bus:
                middleware:
                    - validation
                    - doctrine_transaction
            query.bus:
                middleware:
                    - validation
            event.bus:
                default_middleware: allow_no_handlers
                middleware:
                    - validation
```

Multiple Buses

- Command Bus: no result, async
- Query Bus: wait for result, sync
- Event Bus: separate action from reaction

Each bus has it’s own:

- Handlers
- Set of Middlewares

Why use Messenger for Command Bus?

It’s simple to start.

Why use Messenger for Query Bus?

It’s simple to start.

Why use Messenger for Event Bus?

It’s simple to start.


CQRS ALL THE THINGS!!
memegenerator.net

Transports

- Async/Queued Messages
  - AMQP
  - Doctrine
  - Redis
  - In memory
  - Your own transport (API…)
- Synchronous

Why Messenger?

- Easy to start
- Easy to use different transport mechanisms (rabbitmq, redis, APIs)
- Becomes more powerful accompanied with other Symfony components:
  - Dependency Injection
  - Serializer
  - Event Dispatcher

Why Messenger?

Additional outcomes:

- Audit Logs
- Isolated contexts
- Microservices

Messenger in Business Context

Business Context

We have three departments: warehouse, client relations and the accountants.

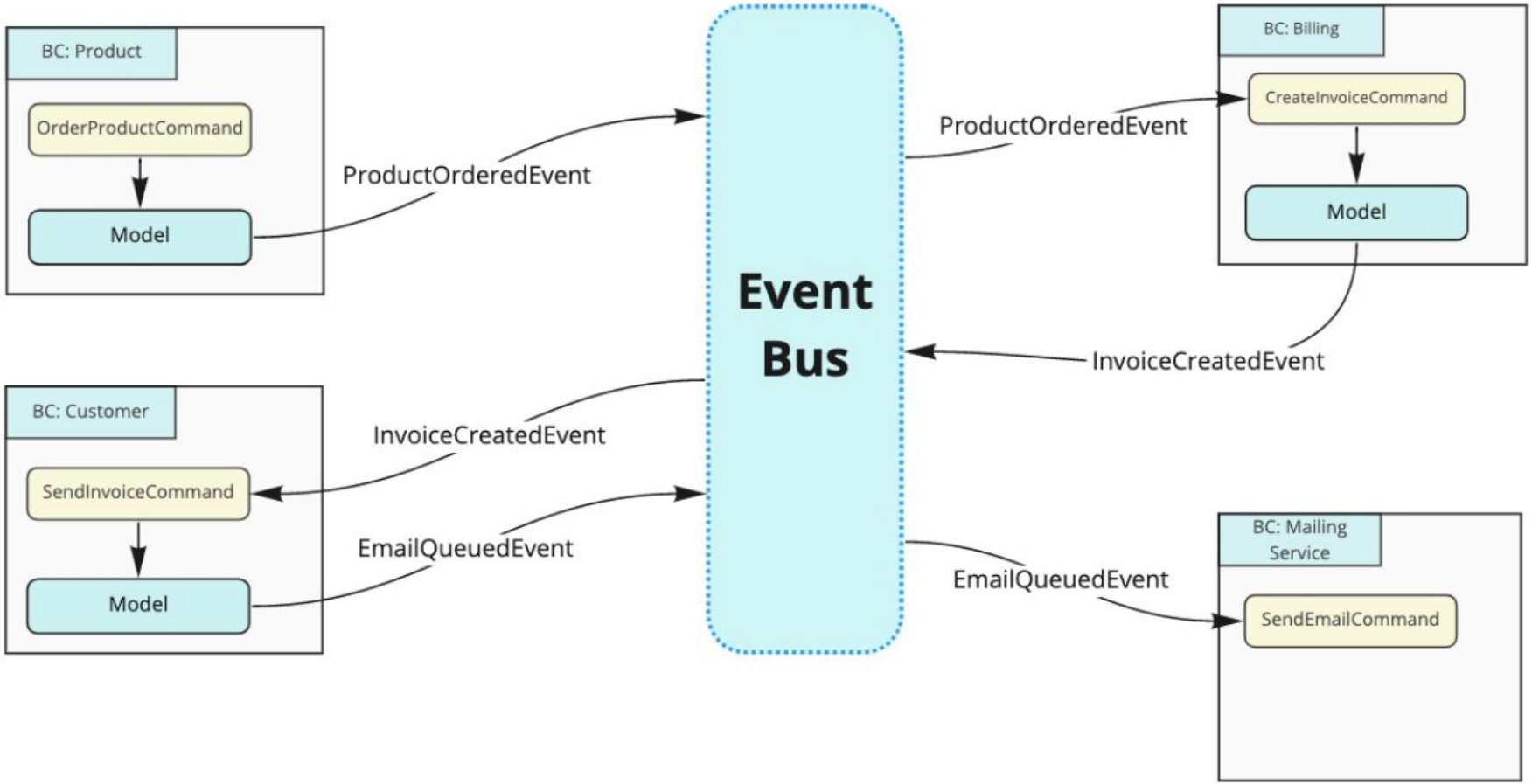
Warehouse manages product and the stock.

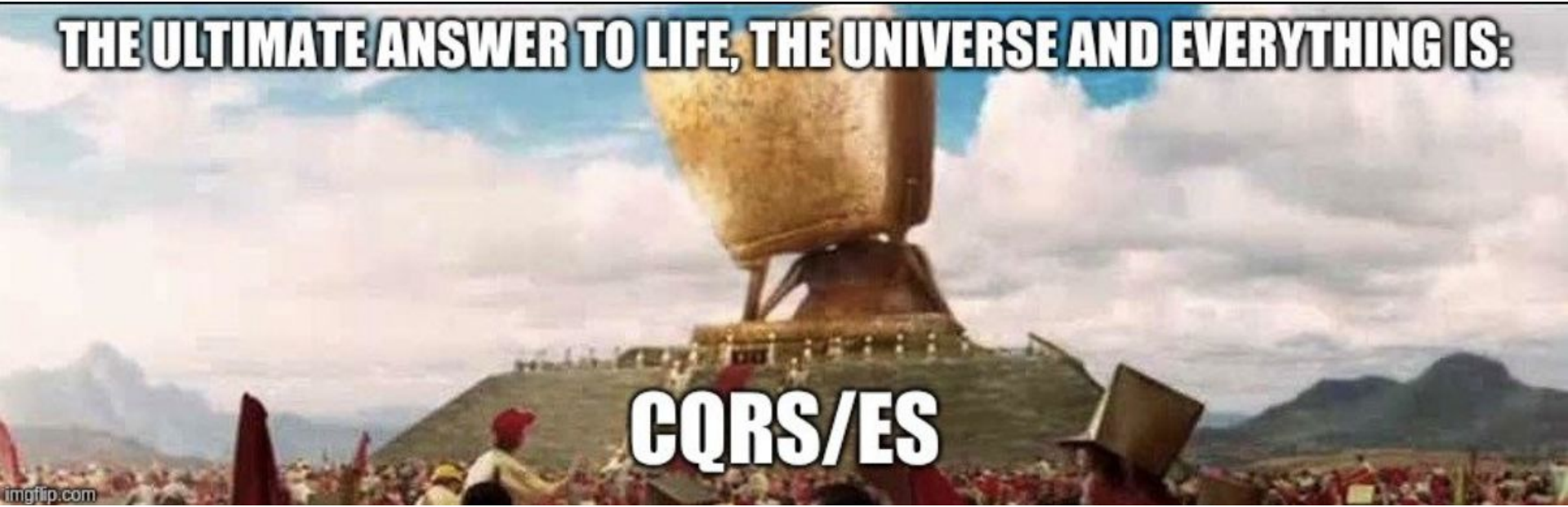
Client relations department - knows everything about their customers.

Accountants - the billing department, takes care of finances.

Problem to solve

We need to send the invoice to the client when he orders the product.


BC: Product
OrderProductCommand
Model
ProductOrderedEvent
Event Bus
BC: Billing
CreateInvoiceCommand
Model
ProductOrderedEvent
InvoiceCreatedEvent
BC: Customer
SendInvoiceCommand
Model
InvoiceCreatedEvent
EmailQueuedEvent
BC: Mailing Service
SendEmailCommand
EmailQueuedEvent


THE ULTIMATE ANSWER TO LIFE, THE UNIVERSE AND EVERYTHING IS:
CQRS/ES
imgflip.com

We are hiring!

15+ open positions

Ačiū!

Klausimai / Idėjos ?

Resources

https://symfony.com/doc/current/components/messenger.html
https://matthiasnoback.nl/2015/01/from-commands-to-events/
https://karoldabrowski.com/blog/query-bus-in-symfony-application/
https://symfonycasts.com/screencast/messenger/install
https://martinfowler.com/bliki/CQRS.html
https://udidahan.com/2009/12/09/clarified-cqrs/