11 Invoicing

We are going to be working on the repository that you have created up to this point. If you skipped ahead, don't worry. You can clone the repository from its Github repository.

```
$  git clone git@github.com:skafandri/symfony-tutorial.git --branch ch10
Cloning into 'symfony-tutorial'...
remote: Counting objects: 1020, done.
remote: Total 1020 (delta 0), reused 0 (delta 0), pack-reused 1020
Receiving objects: 100% (1020/1020), 456.82 KiB | 373.00 KiB/s, done.
Resolving deltas: 100% (518/518), done.
Checking connectivity... done.
```

Don't forget to `composer install`

11.1 Refactoring

Before proceeding we will need to do some refactoring. We will add a **currency** to the Country entity, add **shippingAddress** and **billingAddress** to the Order entity, and update the respective data fixtures.

- Edit **src**/**AppBundle**/**Entity**/**Country.php** and add

```
    /**
     * @var string
     *
     * @ORM\Column(name="currency", type="string", length=3, nullable=false)
     */
    private $currency;

    /**
     * Set currency
     *
     * @param string $currency
     * @return Country
     */
    public function setCurrency($currency)
    {
        $this->currency = $currency;

        return $this;
    }

    /**
     * Get currency
     *
     * @return string
     */
    public function getCurrency()
    {
        return $this->currency;
    }
```

- Edit **src**/**AppBundle**/**DataFixtures**/**ORM**/**CountryFixtures.php** to add the currency and saving references to be used by addresses

```php
<?php

namespace AppBundle\DataFixtures\ORM;

use AppBundle\Entity\Country;

class CountryFixtures extends AbstractDataFixture
{

    private $countries = array(
        'en' => array('england', 'GBP'),
        'es' => array('spain', 'EUR'),
        'fr' => array('france', 'EUR'),
        'it' => array('italy', 'EUR'),
        'ro' => array('romania', 'RON'),
        'tn' => array('tunisia',  'TND'),
    );

    protected function createAndPersistData()
    {
        $countryCount = 0;
        foreach ($this->countries as $code => $country) {
            $countryCount++;
            $countryEntity = new Country();
            $countryEntity->setCode($code)
                          ->setName($country[0])
                          ->setCurrency($country[1]);
            $this->setReference(sprintf('country_%s', $countryCount), $countryEntity);

            $this->manager->persist($countryEntity);
        }
    }

    public function getOrder()
    {
        return 1;
    }

}
```

- Edit **src/AppBundle/DataFixtures/ORM/CustomerFixtures.php** to add creating default addresses for customers
    - Add `use AppBundle\Entity\Address;`
    - Add `$countries = $this->getReferences('country');` after `$index = 0;`
    - Before `$this->manager->persist($customer);` add

```php
$address1 = $this->createAddress($countries[array_rand($countries)]);
$address2 = $this->createAddress($countries[array_rand($countries)]);
$customer->addAddress($address1)->addAddress($address2);

$this->manager->persist($address1);
$this->manager->persist($address2);
```

- Edit **src/AppBundle/Service/OrderService.php** to pick default billing address and shipping address Replace `$order->setCustomer($this->doctrine->getRepository(Customer::REPOSITORY)->find($customerId));` by

```
  /* @var $customer Customer */
  $customer = $this->doctrine->getRepository(Customer::REPOSITORY)->find($customerId);
  $order->setCustomer($customer);
  $addresses = $customer->getAddress();
  $order->setBillingAddress($addresses[0]);
  $order->setShippingAddress($addresses[1]);
```

- Edit **src/AppBundle/Controller/OrderController.php** to use the OrderService for order creation instead of just persisting an entity to the database. Update `createAction` as following

```
/**
 * Creates a new Order.
 *
 */
public function createAction(Request $request) {
    $postOrder = $request->get('appbundle_order');
    $quantities = $request->get('quantity');
    /* @var $orderService \AppBundle\Service\OrderService  */
    $orderService = $this->get(\AppBundle\Service\OrderService::ID);
    $customerId = $postOrder['customer'];
    $productLines = array();

    foreach ($postOrder['productLines'] as $productSaleId => $value) {
        $productLines[] = array(
            'id' => $productSaleId,
            'quantity' => $quantities[$productSaleId]
        );
    }

    $orderId = $orderService->createOrder($customerId, $productLines);


    return $this->redirect(
                    $this->generateUrl('order_show', array('id' => $orderId))
    );
}
```

11.2 Shared session storage

By default, Symfony uses the default PHP session handler, which is disk. We will update our application to use mongoDb as a session storage.

- Edit **app/config/config.yml**
  Change **session.handler_id** from ~ to **session.handler.mongodb**
  Add the following service definition

```
session.handler.mongodb:
    class: Symfony\Component\HttpFoundation\Session\Storage\Handler\MongoDbSessionHandler
    arguments: [@=service('doctrine_mongodb').getConnection().getMongoClient(),
                {database: session, collection: sessions}]
    public: false
```

Now the sessions are stored in mongoDb in **session.sessions**

11.3 Document service

The first feature we are going to implement is generating invoices and saving them into mongoDb.

- Create **src/AppBundle/Document/Document.php**

```php
<?php

namespace AppBundle\Document;

use Doctrine\ODM\MongoDB\Mapping\Annotations as MongoDB;

/**
 * @MongoDB\Document(db="app", collection="documents")
 * @MongoDB\HasLifecycleCallbacks
 */
class Document
{

    const REPOSITORY = 'AppBundle:Document';

    /**
     * @MongoDB\Id
     */
    private $id;

    /**
     * @MongoDB\String
     */
    private $type;

    /**
     * @MongoDB\Increment
     */
    private $number;

    /**
     * @MongoDB\Integer
     */
    private $orderNumber;

    /**
     * @var \DateTime
     * @MongoDB\Date
     */
    private $createDate;

    /**
     * @MongoDB\String
     */
    private $customerName;
```

```php
    /**
     * @MongoDB\String
     */
    private $billingAddress;


    /**
     * @MongoDB\Bin
     */
    private $bodyPdf;

    /**
     * @MongoDB\String
     */
    private $bodyHtml;

    /**
     * @MongoDB\String
     */
    private $currency;

    /**
     * @MongoDB\Integer
     */
    private $total;

    /**
     *
     * @var ProductLine[]
     * @MongoDB\EmbedMany(targetDocument="ProductLine")
     */
    private $productLines = array();

    public function getId() {
        return $this->id;
    }

    public function getType() {
        return $this->type;
    }

    public function getNumber() {
        return $this->number;
    }

    public function getOrderNumber() {
        return $this->orderNumber;
    }

    public function getCreateDate() {
        return $this->createDate;
    }

    public function getCustomerName() {
        return $this->customerName;
    }
```

```php
    public function getBillingAddress() {
        return $this->billingAddress;
    }

    public function getBodyPdf() {
        return $this->bodyPdf;
    }

    public function getBodyHtml() {
        return $this->bodyHtml;
    }

    public function getCurrency() {
        return $this->currency;
    }

    public function getTotal() {
        return $this->total;
    }

    public function getProductLines() {
        return $this->productLines;
    }

    public function setType($type) {
        $this->type = $type;
        return $this;
    }

    public function setOrderNumber($orderNumber) {
        $this->orderNumber = $orderNumber;
        return $this;
    }

    public function setCustomerName($customerName) {
        $this->customerName = $customerName;
        return $this;
    }

    public function setBillingAddress($billingAddress) {
        $this->billingAddress = $billingAddress;
        return $this;
    }

    public function setBodyPdf($bodyPdf) {
        $this->bodyPdf = $bodyPdf;
        return $this;
    }

    public function setBodyHtml($bodyHtml) {
        $this->bodyHtml = $bodyHtml;
        return $this;
    }

    public function setCurrency($currency) {
        $this->currency = $currency;
        return $this;
    }
```

```
    public function setTotal($total) {
        $this->total = $total;
        return $this;
    }


    public function setProductLines(array $productLines) {
        $this->productLines = $productLines;
        return $this;
    }

    public function addProductLine(ProductLine $productLine) {
        $this->productLines[] = $productLine;
    }

    /**
     * @MongoDB\PrePersist
     */
    public function prePersist() {
        $this->createDate = new \DateTime();
    }

}
```

- Create **src/AppBundle/Document/ProductLine.php**

```
<?php

namespace AppBundle\Document;

use Doctrine\ODM\MongoDB\Mapping\Annotations as MongoDB;

/**
 * @MongoDB\EmbeddedDocument
 */
class ProductLine
{

    /**
     * @MongoDB\String
     */
    private $code;

    /**
     * @MongoDB\String
     */
    private $title;

    /**
     * @MongoDB\Integer
     */
    private $quantity;
```

```php
    /**
     * @MongoDB\Integer
     */
    private $price;

    /**
     * @MongoDB\Integer
     */
    private $total;

    public function getCode() {
        return $this->code;
    }

    public function getTitle() {
        return $this->title;
    }

    public function getQuantity() {
        return $this->quantity;
    }

    public function getPrice() {
        return $this->price;
    }

    public function getTotal() {
        return $this->total;
    }

    public function setCode($code) {
        $this->code = $code;
        return $this;
    }

    public function setTitle($title) {
        $this->title = $title;
        return $this;
    }

    public function setQuantity($quantity) {
        $this->quantity = $quantity;
        return $this;
    }

    public function setPrice($price) {
        $this->price = $price;
        return $this;
    }

    public function setTotal($total) {
        $this->total = $total;
        return $this;
    }

}
```

As we did with the confirmation email, we are going to use a Twig template to render the invoice.

- Create **src/AppBundle/Resources/views/Document/Common/base.html.twig**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title></title>
        <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
        <style>
            body {font-size: 12px}
            #body {margin: 20px 20px}
            #footer{
                font-size: 10px;
                color: #888;
                border-top: solid 2px #777
            }
        </style>
    </head>
    <body>
        <table width="800">
            <tr>
                <td>
                    <div id="header">
                        {% block header %}
                            {% include 'AppBundle:Document:Common/header.html.twig'
                                        with {'document': document} %}
                        {% endblock %}
                    </div>
                </td>
            </tr>
            <tr>
                <td>
                    <div id="body">
                        {% block body %}
                            {% include 'AppBundle:Document:Common/body.html.twig'
                                        with {'document': document} %}
                        {% endblock %}
                    </div>
                </td>
            </tr>
            <tr>
                <td>
                    <div id="footer">
                        {% block footer %}
                            {% include 'AppBundle:Document:Common/footer.html.twig' %}
                        {% endblock %}
                    </div>
                </td>
            </tr>
        </table>
    </body>
</html>
```

- Create **src/AppBundle/Resources/views/Document/Common/header.html.twig**

```
<table width="100%">
    <tr>
        <td width="">
            <img src="{{'company.logo'|trans}}">
        </td>
        <td width="200">
            <div>
                {{ ('document.' ~ document.type ~ '.label')|trans|upper}}
            </div>
            <table>
                <tr>
                    <th>{{('document.'~document.type~'.number')|trans|capitalize}}</th>
                    <td>{{document.number}}</td>
                </tr>
                <tr>
                    <th>{{'order.number'|trans|capitalize}}</th>
                    <td>{{document.orderNumber}}</td>
                </tr>
                <tr>
                    <th>{{'date'|trans|capitalize}}</th>
                    <td>{{document.createDate|date('date_format'|trans)}}</td>
                </tr>
                <tr>
                    <th>{{'customer'|trans}}</th>
                    <td>{{document.customerName}}</td>
                </tr>
                <tr>
                    <th>{{'billing_address'|trans}}</th>
                    <td>{{document.billingAddress}}</td>
                </tr>
            </table>
        </td>
    </tr>
</table>
```

- Create **src/AppBundle/Resources/views/Document/Common/body.html.twig**

```
<table width="100%">
    <tr>
        <th width="10%">{{'code'|trans|capitalize}}</th>
        <th width="50%">{{'title'|trans|capitalize}}</th>
        <th width="20%">{{'price'|trans|capitalize}}</th>
        <th width="10%">{{'quantity'|trans|capitalize}}</th>
        <th width="10%">{{'subtotal'|trans|capitalize}}</th>
    </tr>
    {% for productLine in document.productLines %}
        {% include 'AppBundle:Document:Common/product-line.html.twig'
            with {'productLine': productLine, 'currency': document.currency} %}
    {%endfor%}
    <tr>
        <td colspan="2"></td>
        <td>
            {{'total'|trans}}
        </td>
        <td>
            {{ document.total|localizedcurrency(document.currency)}}
        </td>
    </tr>
</table>
```

- Create **src/AppBundle/Resources/views/Document/Common/product-line.html.twig**

```
<tr>
    <td>{{productLine.code}}</td>
    <td>{{productLine.title}}</td>
    <td>{{productLine.price}}</td>
    <td>{{productLine.quantity}}</td>
    <td>{{productLine.total|localizedcurrency(currency)}}</td>
</tr>
```

- Create **src/AppBundle/Resources/views/Document/Common/footer.html.twig**

```
<p>
    {{'company.name'|trans}}
    {{'registration_number'|trans}} {{'company.registration_number'|trans}}
    {{'phone'|trans}} {{'company.phone'|trans}}
</p>
```

- Create **src/AppBundle/Resources/views/Document/invoice.html.twig**

```
{% extends 'AppBundle:Document:Common/base.html.twig' %}
```

We need to enable the Twig Intl extension

- Edit **app/config/config.yml** add the following service definition

```
twig.extension.intl:
    class: Twig_Extensions_Extension_Intl
    public: false
    tags:
        - { name: twig.extension }
```

Let's add the required translation keys

- Edit **src/AppBundle/Resources/translations/messages.en.yml**

```
order:
    status:
        1: new
        10: processing
        11: products missing
        14: products reserved
        15: packaging
        20: delivery started
        29: delivered
        30: cancelled
    number: order number

document:
    invoice:
        label: invoice
        number: invoice number

copyright: copyright
company:
    name: the company
    phone: +123456789
    registration_number: RN999-999

phone: phone
registration_number: registration number
date: date
customer: customer
billing_address: billing address


code: code
title: title
price: price
quantity: quantity
subtotal: subtotal
total: total
date_format: d-m-Y
```

We are going to use wkhtmltopdf to convert HTML to PDF.

Move **src/AppBundle/Service/DocumentService.php** to **src/AppBundle/Service/Document/DocumentService.php** and let's implement it logic: generating invoice as PDF and persisting to mongoDb

```php
<?php

namespace AppBundle\Service\Document;

use AppBundle\Document\Document;
use AppBundle\Document\ProductLine;
use AppBundle\Entity\Order;
use AppBundle\Event\Order\OrderEvent;
use Doctrine\Bundle\MongoDBBundle\ManagerRegistry;
use Doctrine\ODM\MongoDB\DocumentManager;
use Symfony\Component\EventDispatcher\EventDispatcherInterface;
use Symfony\Component\Process\Process;
use Symfony\Component\Templating\EngineInterface as Templating;

class DocumentService
{

    const ID = 'app.document';

    /**
     *
     * @var Templating
     */
    private $twigEngine;

    /**
     *
     * @var EventDispatcherInterface
     */
    private $eventDispatcher;

    /**
     *
     * @var DocumentManager
     */
    private $documentManager;

    public function __construct(
        Templating $twigEngine,
        EventDispatcherInterface $eventDispatcher,
        ManagerRegistry $documentManager
    ) {
        $this->twigEngine = $twigEngine;
        $this->eventDispatcher = $eventDispatcher;
        $this->documentManager = $documentManager->getManager();
    }

    public function generateInvoice(Order $order) {
        $templateName = 'AppBundle:Document:invoice.html.twig';
        $document = $this->createDocumentFromOrder($order);
        $html = $this->twigEngine->render($templateName, array('document' => $document));
        $document->setBodyHtml($html);
        $document->setBodyPdf($this->convertHtmlToPdf($html));
        $this->documentManager->persist($document);
        $this->documentManager->flush();
```

```
        $this->eventDispatcher->dispatch(
                OrderEvent::INVOICE_GENERATED, new OrderEvent($order)
        );
    }

    private function createDocumentFromOrder(Order $order) {
        $document = new Document();
        $billingAddress = $order->getBillingAddress();
        $productLines = $this->createProductLinesFromOrder($order);
        $total = 0;
        foreach ($productLines as $productLine) {
            $total += $productLine->getTotal();
        }
        $document->setBillingAddress($billingAddress->getName())
                ->setCurrency($billingAddress->getCountry()->getCurrency())
                ->setCustomerName($order->getCustomer()->getContact()->getName())
                ->setOrderNumber($order->getId())
                ->setProductLines($productLines)
                ->setTotal($total)
                ->setType('invoice');

        return $document;
    }

    private function createProductLinesFromOrder(Order $order) {
        $productLines = array();
        foreach ($order->getProductLines() as $orderProductLine) {
            $code = $orderProductLine->getProductSale()->getProduct()->getCode();
            $title = $orderProductLine->getProductSale()->getProduct()->getTitle();
            $price = $orderProductLine->getProductSale()->getPrice();
            $quantity = $orderProductLine->getQuantity();

            $productLine = new ProductLine();
            $productLine->setCode($code)
                    ->setPrice($price)
                    ->setQuantity($quantity)
                    ->setTitle($title)
                    ->setTotal($quantity * $price);
            $productLines[] = $productLine;
        }
        return $productLines;
    }

    private function convertHtmlToPdf($html) {
        $process = new Process(sprintf("echo '%s' | wkhtmltopdf - -", $html));
        $process->run();
        return $process->getOutput();
    }

}
```

We need to update the document service definition, edit **src/AppBundle/Resources/config/services.yml**

```
app.document:
    class: AppBundle\Service\Document\DocumentService
    arguments: [@templating, @event_dispatcher, @doctrine_mongodb]
```

Now we will update the OrderListener to generate an invoice when the products are reserved.

- Edit **src/AppBundle/Event/Listener/OrderListener.php**
  add `use AppBundle\Service\Document\DocumentService;`
  add $documentService member

  ```
  /**
  *
  * @var DocumentService
  */
  private $documentService;
  ```

  Update the constructor

  ```
  public function __construct(
    WarehouseService $warehouseService,
        DeliveryService $deliveryService,
        DocumentService $documentService
  ) {
    $this->warehouseService = $warehouseService;
    $this->deliveryService = $deliveryService;
    $this->documentService = $documentService;
  }
  ```

  Add to the end of `onProductsReserved`

  ```
  $this->documentService->generateInvoice($event->getOrder());
  ```

Update the OrderListener service definition in **src/AppBundle/Resources/config/listeners.yml** and add **@app.document** as third argument.

11.4 View invoice in order View

We want to be able to view the generated invoice from the order view

- Edit **src/AppBundle/Service/Document/DocumentService.php**
  Add `use AppBundle\Exception\Document\DocumentNotFoundException;`
  Add `getInvoiceHtml` method

```
public function getInvoiceHtml($orderNumber) {
  $repository = $this->documentManager->getRepository(Document::REPOSITORY);
  $document = $repository->findOneBy(
          array('type' => 'invoice', 'orderNumber' => (int) $orderNumber)
  );
  if (!$document) {
      throw new DocumentNotFoundException();
  }
  return $document->getBodyHtml();
}
```

- Create **src/AppBundle/Exception/Document/DocumentNotFoundException.php**

```
<?php

namespace AppBundle\Exception\Document;

use AppBundle\Exception\AppException;

class DocumentNotFoundException extends AppException
{

}
```

- Edit **src/AppBundle/Controller/OrderController.php**
  Add `use AppBundle\Service\Document\DocumentService`
  Add `viewInvoiceAction`

```
public function viewInvoiceAction($orderId) {
  /* @var $documentService DocumentService */
  $documentService = $this->get(DocumentService::ID);
  $html = $documentService->getInvoiceHtml($orderId);

  return new \Symfony\Component\HttpFoundation\Response($html);
}
```

- Edit **src/AppBundle/Resources/config/routing/order.yml** add the route

```
order_invoice_view:
    path:     /{orderId}/invoice/view
    defaults: { _controller: "AppBundle:Order:viewInvoice" }
```

- Edit **src/AppBundle/Resources/views/Order/show.html.twig** after the records table, insert

```
<div>
    <a class="button" target="_blank"
       accesskey=""href="{{ path('order_invoice_view', { 'orderId': entity.id }) }}">
        view invoice
    </a>
</div>
```

11.5 RabbitMq

RabbitMq is a message broker software that implements the Advanced Message Queuing Protocol (AMQP).

For installation options, please visit https://www.rabbitmq.com/download.html

For more detailed documentation, please visit https://www.rabbitmq.com/documentation.html

First thing, we need to install the RabbitMq symfony bundle.

- Edit **composer.json** and add the following requirement `"oldsound/rabbitmq-bundle": "1.7.0"`. Run `composer update`

To enable the new bundle, edit **app/AppKernel.php** and add `new OldSound\RabbitMqBundle\OldSoundRabbitMqBundle()` to the bundles array.

Second, we need to configure the RabbitMq bundle

- Create **app/config/rabbitmq.yml**

```
old_sound_rabbit_mq:
    connections:
        default:
            host:     'localhost'
            port:     5672
            user:     'guest'
            password: 'guest'
            vhost:    '/'
            lazy:     false
            connection_timeout: 3
            read_write_timeout: 3
            keepalive: false
    producers:
        document:
            connection:       default
            exchange_options: {name: 'document', type: direct}
    consumers:
        document.invoice:
            connection:       default
            exchange_options: {name: 'document', type: direct}
            queue_options:    {name: 'document.invoice', routing_keys: [document.invoice]}
            callback:         app.document
```

- Edit **app/config/config.yml** and add `- { resource: rabbitmq.yml }` to the imports list

Done. We can now update our code to do the following:
When calling generateInvoice method, we will just create the document in mongoDb and *publish* the document Id to a queue. A consumer will fetch the document based on it's Id, render the invoice HTML, and generates the PDF file.

- Create **src/AppBundle/Event/Document/DocumentEvent.php**

```php
<?php

namespace AppBundle\Event\Document;

use AppBundle\Document\Document;
use AppBundle\Event\LoggableEventInterface;
use Symfony\Component\EventDispatcher\Event;

class DocumentEvent extends Event implements LoggableEventInterface
{

    const INVOICE_GENERATE_START = 'document.invoice.generate.start';
    const INVOICE_GENERATE_FINISH = 'document.invoice.generate.finish';

    /**
     *
     * @var Document
     */
    private $document;

    public function __construct(Document $document) {
        $this->document = $document;
    }

    public function getLogContext() {
        return array(
            'documentId' => $this->document->getId(),
            'orderNumber' => $this->document->getOrderNumber()
        );
    }

}
```

- Edit **src/AppBundle/Service/Document/DocumentService.php**
  Add the uses:

```php
use AppBundle\Event\Document\DocumentEvent;
use OldSound\RabbitMqBundle\RabbitMq\Producer;
use PhpAmqpLib\Message\AMQPMessage;
```

Add the member documentProducer

```php
/**
*
* @var Producer
*/
private $documentProduce
```

Update the constructor to accept a producer:

```
public function __construct(
  Templating $twigEngine,
  EventDispatcherInterface $eventDispatcher,
  ManagerRegistry $documentManager,
  Producer $documentProducer
) {
  $this->twigEngine = $twigEngine;
  $this->eventDispatcher = $eventDispatcher;
  $this->documentManager = $documentManager->getManager();
  $this->documentProducer = $documentProducer;
}
```

Update generateInvoice to just create the document and publish it's Id to the queue

```
public function generateInvoice(Order $order) {
  $document = $this->createDocumentFromOrder($order);
  $this->eventDispatcher->dispatch(
          DocumentEvent::INVOICE_GENERATE_START, new DocumentEvent($document)
  );
  $this->documentManager->persist($document);
  $this->documentManager->flush();
  $this->documentProducer->publish($document->getId(), 'document.invoice');
}
```

Create execute method

```
public function execute(AMQPMessage $amqpMessage) {
  $documentId = $amqpMessage->body;
  $repository = $this->documentManager->getRepository(Document::REPOSITORY);
  $document = $repository->find($documentId);
  $templateName = 'AppBundle:Document:invoice.html.twig';
  $html = $this->twigEngine->render($templateName, array('document' => $document));
  $document->setBodyHtml($html);
  $document->setBodyPdf($this->convertHtmlToPdf($html));
  $this->documentManager->flush();

  $this->eventDispatcher->dispatch(
          DocumentEvent::INVOICE_GENERATE_FINISH, new DocumentEvent($document)
  );

  return true;
}
```

We need to update the document service definition, edit **src/AppBundle/Resources/config/services.yml** and add `@old_sound_rabbit_mq.document_producer` as last argument to *app.document* service.

Now if you create a new order, no invoice will be generated. To generate it, run the command:

```
app/console rabbitmq:consumer document.invoice -m 1 -w
```

11.6 Exercices

- Create a service AppBundle\Service\UtilService with a method

```
public function getLock($lockName, $timeout){

}
```

Use Mysql **GET_LOCK()** function to achieve the locking mechanism

- When the invoice is generated, email it to the client.
- Decouple the email sending mechanism to use a producer/consumer strategy.
- Create a warning mechanism that will send an email to an administrator when an invoice body was not generated after X minutes from it's creation. (the administrator's email address and the X minutes are parameters)