9 Communication service

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 ch8
Cloning into 'symfony-tutorial'...
remote: Counting objects: 838, done.
remote: Compressing objects: 100% (93/93), done.
remote: Total 838 (delta 27), reused 0 (delta 0), pack-reused 737
Receiving objects: 100% (838/838), 433.96 KiB | 319.00 KiB/s, done.
Resolving deltas: 100% (405/405), done.
Checking connectivity... done.
```

Don't forget to `composer install`

Make sure the Elasticsearch server and Kibana we installed last time are running, so we can easily view the generated logs.

9.1 Extra data fixtures

Before proceeding with further functionality implementation, we will add some data fixtures.

- Edit **src/AppBundle/DataFixtures/ORM/AbstractDataFixture.php**

```php
<?php

namespace AppBundle\DataFixtures\ORM;

use Doctrine\Common\DataFixtures\AbstractFixture;
use Doctrine\Common\DataFixtures\OrderedFixtureInterface;
use Doctrine\Common\Persistence\ObjectManager;

abstract class AbstractDataFixture extends AbstractFixture
                                    implements OrderedFixtureInterface
{

    /**
     *
     * @var ObjectManager
     */
    protected $manager;

    public function load(ObjectManager $manager)
    {
        $this->manager = $manager;
        $this->createAndPersistData();
        $this->manager->flush();
    }

    abstract protected function createAndPersistData();

    protected function getReferences($prefix)
    {
        $entities = array();
        for ($i = 1; true; $i++) {
            try {
                $entities[] = $this->getReference(sprintf('%s_%s', $prefix, $i));
            } catch (\OutOfBoundsException $exception) {
                break;
            }
        }

        return $entities;
    }

}
```

- Edit **src/AppBundle/DataFixtures/ORM/CategoryFixtures.php**

```php
<?php

namespace AppBundle\DataFixtures\ORM;

use AppBundle\Entity\Category;

class CategoryFixtures extends AbstractDataFixture
{

    private $categoriesCount = 0;
    private $categories = array(
        'computers' => array('servers', 'desktop',
                             'laptops', 'components', 'periferals'),
        'phones and tablets' => array('phones', 'tablets', 'accessories'),
        'appliances' => array('coffe machines',
                              'washing machines', 'blenders', 'juicers'),
        'video games' => array('consoles', 'games', 'accessories')
    );

    protected function createAndPersistData()
    {
        foreach ($this->categories as $parent => $children) {
            $parentCategory = $this->createAndPersistCategory($parent);
            foreach ($children as $label) {
                $this->createAndPersistCategory($label, $parentCategory);
            }
        }
    }

    private function createAndPersistCategory($label, $parentCategory = null)
    {
        $this->categoriesCount ++;
        $category = new Category();
        $category->setLabel($label)->setParentCategory($parentCategory);
        $this->manager->persist($category);

        $this->setReference(
            sprintf('category_%s', $this->categoriesCount), $category
        );

        return $category;
    }

    public function getOrder()
    {
        return 1;
    }

}
```

- Edit **src/AppBundle/DataFixtures/ORM/ProductFixtures.php**

```php
<?php

namespace AppBundle\DataFixtures\ORM;

use AppBundle\Entity\Category;
use AppBundle\Entity\Product;
use Symfony\Component\DependencyInjection\ContainerAwareInterface;
use Symfony\Component\DependencyInjection\ContainerInterface;

class ProductFixtures extends AbstractDataFixture implements ContainerAwareInterface
{

    private $productsCount = 0;
    private $productsPerCategory;

    private function createAndPersistProducts(Category $category)
    {
        for ($i = 1; $i <= $this->productsPerCategory; $i++) {
            $this->productsCount++;
            $product = new Product();
            $product->setCode(sprintf('code_%s_%s', $category->getId(), $i))
                    ->setTitle(
                      sprintf('title %s %s %s', $category->getId(), $i, uniqid())
                    )
                    ->setDescription(sprintf('product description %s', $i));
            $category->addProduct($product);
            $this->manager->persist($product);
            $this->setReference(
                sprintf('product_%s', $this->productsCount), $product
            );
        }
    }

    protected function createAndPersistData()
    {
        foreach ($this->getReferences('category') as $category) {
            $this->createAndPersistProducts($category);
        }
    }

    public function setContainer(ContainerInterface $container = null)
    {
        $fixturesConfig = $container->getParameter('fixtures');
        $this->productsPerCategory = $fixturesConfig['products_per_category'];
    }

    public function getOrder()
    {
        return 2;
    }
}
```

- Create **src/AppBundle/DataFixtures/ORM/AccountFixtures.php**

```php
<?php

namespace AppBundle\DataFixtures\ORM;

use AppBundle\Entity\Account;

class AccountFixtures extends AbstractDataFixture
{

    const ACCOUNTS_COUNT = 10;

    protected function createAndPersistData()
    {
        for ($i = 1; $i <= self::ACCOUNTS_COUNT; $i++) {
            $account = new Account();
            $account->setActive(true);
            $account->setEmail(sprintf('email%s@email.com', $i));
            $account->setPassword('password');
            $this->manager->persist($account);
            $this->setReference(sprintf('account_%s', $i), $account);
        }
    }

    public function getOrder()
    {
        return 1;
    }

}
```

- Create **src/AppBundle/DataFixtures/ORM/ContactFixtures.php**

```php
<?php

namespace AppBundle\DataFixtures\ORM;

use AppBundle\Entity\Contact;

class ContactFixtures extends AbstractDataFixture
{

    const CONTACTS_COUNT = 10;

    protected function createAndPersistData()
    {
        for ($i = 1; $i <= self::CONTACTS_COUNT; $i++) {
            $contact = new Contact();
            $contact->setName(sprintf('contact %s', $i));
            $contact->setEmail(sprintf('contact_email%s@email.com', $i));
            $contact->setMobilePhone(rand(1111111111, 9999999999));
            $contact->setPhone(rand(1111111111, 9999999999));
            $this->manager->persist($contact);
            $this->setReference(sprintf('contact_%s', $i), $contact);
        }
    }

    public function getOrder()
    {
        return 1;
    }

}
```

- Create **src/AppBundle/DataFixtures/ORM/CountryFixtures.php**

```php
<?php

namespace AppBundle\DataFixtures\ORM;

use AppBundle\Entity\Country;

class CountryFixtures extends AbstractDataFixture
{

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

    protected function createAndPersistData()
    {
        foreach ($this->countries as $countryCode => $name) {
            $country = new Country();
            $country->setCode($countryCode)->setName($name);
            $this->manager->persist($country);
        }
    }

    public function getOrder()
    {
        return 1;
    }

}
```

- Create **src/AppBundle/DataFixtures/ORM/CustomerFixtures.php**

```php
<?php
namespace AppBundle\DataFixtures\ORM;
use AppBundle\Entity\Customer;

class CustomerFixtures extends AbstractDataFixture
{
    protected function createAndPersistData()
    {
        $index = 0;
        foreach ($this->getReferences('account') as $account) {
            $index++;
            $contact = $this->getReference(sprintf('contact_%s', $index));
            $customer = new Customer();
            $customer->setAccount($account)->setContact($contact);
            $this->manager->persist($customer);
        }
    }
    public function getOrder()
    {
        return 2;
    }
}
```

- Create **src/AppBundle/DataFixtures/ORM/ProductSaleFixtures.php**

```php
<?php
namespace AppBundle\DataFixtures\ORM;
use AppBundle\Entity\ProductSale;

class ProductSaleFixtures extends AbstractDataFixture
{
    protected function createAndPersistData()
    {
        foreach ($this->getReferences('product') as $product) {
            $productSale = new ProductSale();
            $productSale->setProduct($product)->setActive(rand(0, 1));
            $productSale->setPrice(rand(100, 10000));
            $randomDate = time() + rand(-100000, 100000);
            $startDate = new \DateTime();
            $startDate->setTimestamp($randomDate);
            $endDate = new \DateTime();
            $endDate->setTimestamp($randomDate + rand(100000, 1000000));
            $productSale->setStartDate($startDate)->setEndDate($endDate);
            $this->manager->persist($productSale);
        }
    }
    public function getOrder()
    {
        return 3;
    }
}
```

- Create **src/AppBundle/DataFixtures/ORM/ProductStockFixtures.php**

```php
<?php

namespace AppBundle\DataFixtures\ORM;

use AppBundle\Entity\ProductStock;

class ProductStockFixtures extends AbstractDataFixture
{

    protected function createAndPersistData()
    {
        foreach ($this->getReferences('product') as $product) {
            $productStock = new ProductStock();
            $productStock->setProduct($product)->setQuantity(rand(0, 1000));
            $warehouse = $this->getReference(sprintf('warehouse_%s', rand(1, 10)));
            $productStock->setWarehouse($warehouse);
            $this->manager->persist($productStock);
        }
    }

    public function getOrder()
    {
        return 3;
    }
}
```

- Create **src/AppBundle/DataFixtures/ORM/VendorFixtures.php**

```php
<?php

namespace AppBundle\DataFixtures\ORM;

use AppBundle\Entity\Vendor;

class VendorFixtures extends AbstractDataFixture
{

    protected function createAndPersistData()
    {
        for ($i = 1; $i <= 10; $i++) {
            $vendor = new Vendor();
            $vendor->setName(sprintf('vendor_%s', $i));
            $this->manager->persist($vendor);
        }
    }

    public function getOrder()
    {
        return 1;
    }
}
```

- Create **src/AppBundle/DataFixtures/ORM/WarehouseFixtures.php**

```php
<?php

namespace AppBundle\DataFixtures\ORM;

use AppBundle\Entity\Warehouse;

class WarehouseFixtures extends AbstractDataFixture
{

    protected function createAndPersistData()
    {
        for ($i = 1; $i <= 10; $i++) {
            $warehouse = new Warehouse();
            $warehouse->setName(sprintf('warehouse_%s', $i));
            $warehouse->setAddress(sprintf('warehouse address %s', $i));
            $this->manager->persist($warehouse);
            $this->setReference(sprintf('warehouse_%s', $i), $warehouse);
        }
    }

    public function getOrder()
    {
        return 1;
    }
}
```

Done, now we can load the new fixtures into the database

```
$ app/console doctrine:schema:drop --force
Dropping database schema...
Database schema dropped successfully!

$ app/console doctrine:schema:create
ATTENTION: This operation should not be executed in a production environment.

Creating database schema...
Database schema created successfully!

$ app/console doctrine:fixtures:load
Careful, database will be purged. Do you want to continue Y/N ?y
  > purging database
  > loading [1] AppBundle\DataFixtures\ORM\ContactFixtures
  > loading [1] AppBundle\DataFixtures\ORM\VendorFixtures
  > loading [1] AppBundle\DataFixtures\ORM\AccountFixtures
  > loading [1] AppBundle\DataFixtures\ORM\WarehouseFixtures
  > loading [1] AppBundle\DataFixtures\ORM\CategoryFixtures
  > loading [1] AppBundle\DataFixtures\ORM\CountryFixtures
  > loading [2] AppBundle\DataFixtures\ORM\CustomerFixtures
  > loading [2] AppBundle\DataFixtures\ORM\ProductFixtures
  > loading [3] AppBundle\DataFixtures\ORM\ProductSaleFixtures
  > loading [3] AppBundle\DataFixtures\ORM\ProductStockFixtures
```

9.2 Send confirmation email to customer

We are going to implement the emails sending logic. We will use a mixture of Twig templates and translation files to render the email HTML.

First, we have a small bug to fix.

- Edit **src/AppBundle/Service/Communication/EmailService.php** and change
  `$provider = $this->providerIndex[$this->providerIndex];`
  to
  `$provider = $this->providers[$this->providerIndex];`

- Create **src/AppBundle/Resources/views/Templates/Email/base.html.twig**

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8" />
        <title>
            {% block title %}
                {% if title is defined %}
                    {{title|trans}}
                {% endif %}
            {% endblock %}
        </title>
    </head>
    <body>
        {% block body %}
            <div id="container" style="width: 600px" >
                <div id="header">
                    {% block header %}
                        <h1> {{ 'company_name'|trans|capitalize }}</h1>
                    {% endblock %}
                </div>
                <div id="content">
                    {% block content %}
                    {% endblock %}
                </div>
                <div id="footer" style="font-size: 10px">
                    {% block footer %}
                        <hr/>
                        {{ 'copyright'|trans|capitalize }} {{ "now"|date("Y") }}
                        {{ 'company_name'|trans}}
                    {% endblock %}
                </div>
            </div>
        {% endblock %}
    </body>
</html>
```

- Create **src/AppBundle/Resources/views/Templates/Email/confirmation.html.twig**

```
{% extends 'AppBundle:Templates:Email/base.html.twig' %}

{% block content %}
    {{ 'confirmation.text'|trans(
      {'%customerName%': customerName, '%orderNumber%': orderNumber},
      'email'
      ) }}
{% endblock %}
```

- 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

copyright: copyright
company_name: the company
```

- Create **src/AppBundle/Resources/translations/email.en.yml**

```
confirmation:
    text: Thank you %customerName%, your order number
          %orderNumber% is being processed.
    subject: Order confirmation
    from: orders@site.com
```

We forgot to add a **from** field to our email Message. Edit **src/AppBundle/Communication/Email/Message.php** to add the **from** member and it's accessors.

```
private $from;
...
public function getFrom()
{
    return $this->from;
}

public function setFrom($from)
{
    $this->from = $from;
    return $this;
}
```

Let's create some new events related to email sending

- Create **src/AppBundle/Event/Communication/Email/EmailEvent.php**

```php
<?php

namespace AppBundle\Event\Communication\Email;

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

class EmailEvent extends Event implements LoggableEventInterface
{
    const BEFORE_SEND = 'email.before_send';
    const ERROR_TEMPORARY = 'email.error_temporary';
    const ERROR_PERMANENT = 'email.error_permanent';
    const SENT = 'email.sent';

    private $type;
    private $emailAddress;
    private $arguments;

    public function __construct($type, $emailAddress, $arguments)
    {
        $this->type = $type;
        $this->emailAddress = $emailAddress;
        $this->arguments = $arguments;
    }

    public function getLogContext()
    {
        return array(
            'type' => $this->type,
            'emailAddress' => $this->emailAddress,
            'arguments' => $this->arguments,
        );
    }

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

    public function getEmailAddress()
    {
        return $this->emailAddress;
    }

    public function getArguments()
    {
        return $this->arguments;
    }
}
```

- Create **src/AppBundle/Event/Communication/Email/EmailSendingEvent.php**

```php
<?php

namespace AppBundle\Event\Communication\Email;

use AppBundle\Communication\Email\Message;

class EmailSendingEvent extends EmailEvent
{

    /**
     *
     * @var Message
     */
    private $message;

    public function __construct($type, $arguments, Message $message)
    {
        parent::__construct($type, $message->getTo(), $arguments);
        $this->message = $message;
    }

    public function getLogContext()
    {
        $context = array(
            'from' => $this->message->getFrom(),
            'subject' => $this->message->getSubject(),
            'body' => $this->message->getMessage()
        );
        return array_merge(parent::getLogContext(), $context);
    }

    public function getMessage()
    {
        return $this->message;
    }

}
```

We will update the CommunicationService to require 3 new dependencies, a template renderer, a translator, and an event dispatcher.

- Edit **src/AppBundle/Service/Communication/CommunicationService.php**

```php
<?php

namespace AppBundle\Service\Communication;

use AppBundle\Communication\Email\Message;
use AppBundle\Event\Communication\Email\EmailEvent;
use AppBundle\Event\Communication\Email\EmailSendingEvent;
use Symfony\Component\EventDispatcher\EventDispatcherInterface;
use Symfony\Component\Templating\EngineInterface as Templating;
use Symfony\Component\Translation\TranslatorInterface as Translator;

class CommunicationService
{

    const ID = 'app.communication';

    /**
     *
     * @var EmailService
     */
    private $emailService;

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

    /**
     *
     * @var Translator
     */
    private $translator;

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

    public function __construct(
            EmailService $emailService,
            Templating $twigEngine,
            Translator $translator,
            EventDispatcherInterface $eventDispatcher
    )
    {
        $this->emailService = $emailService;
        $this->twigEngine = $twigEngine;
        $this->translator = $translator;
        $this->eventDispatcher = $eventDispatcher;
    }
```

```
    public function sendConfirmationEmail(
        $emailAddress,
        $name, $orderNumber,
        $locale = 'en'
    )
    {
        $arguments = array('customerName' => $name, 'orderNumber' => $orderNumber);
        return $this->sendEmail('confirmation', $emailAddress, $arguments, $locale);
    }

    public function sendDeliveryEmail($emailAddress, $orderNumber)
    {

    }

    public function sendInvoice($emailAddress, $orderNumber)
    {

    }

    public function sendSatisfactionSurvey($emailAddress, $orderNumber)
    {

    }

    public function sendEmail($type, $emailAddress, $arguments, $locale = 'en')
    {
        $this->eventDispatcher->dispatch(
            EmailEvent::BEFORE_SEND, new EmailEvent($type, $emailAddress, $arguments)
        );
        $this->translator->setLocale($locale);
        $message = $this->constructEmailMessage($type, $emailAddress, $arguments);
        $status = $this->emailService->send($message);
        $this->dispatchEmailSendingEvent($type, $arguments, $message, $status);
        return $status;
    }

    private function dispatchEmailSendingEvent($type, $arguments, $message, $status)
    {
        $event = new EmailSendingEvent($type, $arguments, $message);
        $this->eventDispatcher->dispatch(EmailSendingEvent::SENT, $event);
    }

    private function renderTempalate($type, $arguments)
    {
        $templateName = sprintf('AppBundle:Templates:Email/%s.html.twig', $type);
        return $this->twigEngine->render($templateName, $arguments);
    }

    private function getSubject($type, $arguments)
    {
        $translationKey = sprintf('%s.subject', $type);
        return $this->translator->trans($translationKey, $arguments, 'email');
    }
```

```
    private function getFrom($type)
    {
        $translationKey = sprintf('%s.from', $type);
        return $this->translator->trans($translationKey, array(), 'email');
    }

    private function constructEmailMessage($type, $emailAddress, $arguments)
    {
        $message = new Message();
        $message->setTo($emailAddress);
        $message->setMessage($this->renderTempalate($type, $arguments));
        $message->setSubject($this->getSubject($type, $arguments));
        $message->setFrom($this->getFrom($type));
        return $message;
    }

}
```

Now we need to update the service defintion to inject the new dependencies.

- Edit **src/AppBundle/Resources/config/services.yml** and upadte **app.communication.arguments** `arguments: [@app.email, @templating, @translator, @event_dispatcher]`

Since we changed *sendConfirmationEmail* in CommunicationServie, we need to change the call in OrderCommunicationListener

- Edit **src/AppBundle/Event/Listener/OrderCommunicationListener.php**
  Update *onAfterCreate* and add *getCustomerName* method

```
public function onAfterCreate(OrderEvent $event)
{
    $emailAddress = $this->getEmailAddress($event);
    $customerName = $this->getCustomerName($event);
    $this->communicationService->sendConfirmationEmail(
            $emailAddress, $customerName, $event->getOrder()->getId()
    );
}

private function getCustomerName(OrderEvent $event)
{
    return $event->getOrder()->getCustomer()->getContact()->getName();
}
```

Done. Try to create an order through the JSON-RPC API. You should see new logs about the email sending.

```
[] app.INFO: email.before_send {"type":"confirmation","emailAddress":
"contact_email1@email.com","arguments":{"customerName":"contact1","orderNumber":1}}[]
[] app.INFO: email.sent {"type":"confirmation","emailAddress":
"contact_email1@email.com","arguments":{"customerName":"contact1","orderNumber":1},
"from":"orders@site.com","subject": "Order confirmation","body":"<!DOCTYPE html>\n
<html>\n  <head>\n<meta charset=\"UTF-8\" />\n     <title>\n  </title>\n </head>\n
  <body>\n<div id=\"container\" style=\"width: 600px\" >\n   <div id=\"header\">\n
<h1> The company</h1>\n</div>\n <div id=\"content\">\nThank you contact1,
your order number 1 is being processed.\n </div>\n
<div id=\"footer\" style=\"font-size: 10px\">\n<hr/>\n
 Copyright 2015 the company\n     </div>\n    </div>\n  </body>\n</html>"} []
```

9.3 MongoDB

Previously, we logged an entry that looks like:

```
{
      "type": "confirmation",
      "emailAddress": "contact_email1@email.com",
      "arguments": {
        "customerName": "contact 1",
        "orderNumber": 29
      },
      "from": "orders@site.com",
      "subject": "Order confirmation",
      "body": "..."
    }
```

We want to store this data in a searcheable format and eventually display it to an end user. An RDMS would be a good option for us, MySQL for instance since we are already using it, but..

- The **body** column can hold large data (is a HTML page)
- We don't know the possible keys for the **arguments** column, so we will need to update it quite frequently.
- We will eventually include other columns more than (type, emailAddress, from...)
- We definitely don't want any performance donwside

Our rescue to this situation is noSql databases, a document-oriented databse would be even better. We will use MongoDB as data storage for those email related datasets.

Let's start by installing a MongoDB server and see it working. The installation procedure is very simple.

```
$ sudo apt-get install mongodb-server
$ mkdir ~/mongo
$ mongod --dbpath ~/mongo --rest
```

> If you get an error **locale::facet::_S_create_c_locale name not valid** means that MongoDB is unable to work with your operating system's current locale. Adjusting your locale to C, should solve it: `$ export LC_ALL=C`

And start the server again. If everything is OK, you should be able to access the management interface at http://127.0.0.1:28017/

For more installation options, please visit http://docs.mongodb.org/manual/installation/

We can go back to our application now, and implement persisting this data to MongoDB.

First, we need to load DoctrineMongoDBBundle.

- Edit **app/AppKernel.php** and add
  `new Doctrine\Bundle\MongoDBBundle\DoctrineMongoDBBundle(),`
  to the list of loaded bundles.

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

```
doctrine_mongodb:
    connections:
        default:
            server: mongodb://localhost:27017
            options: {}
    document_managers:
        default:
            auto_mapping: true
```

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

Working with documents is not very different than working with entities, let's create the email document.

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

```
<?php

namespace AppBundle\Document;

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

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

    const REPOSITORY = 'AppBundle:Email';

    const STATUS_SENT = 1;
    const STATUS_TEMPORARY_ERROR = 2;
    const STATUS_PERMANENT_ERROR = 3;
```

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

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

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

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

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

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

    /**
     * @MongoDB\Int
     */
    private $status;

    /**
     * @MongoDB\Hash
     */
    private $arguments;

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

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

    public function getEmailAddress()
    {
        return $this->emailAddress;
    }
```

```
    public function getFrom()
    {
        return $this->from;
    }

    public function getSubject()
    {
        return $this->subject;
    }

    public function getBody()
    {
        return $this->body;
    }

    public function getArguments()
    {
        return $this->arguments;
    }

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

    public function setEmailAddress($emailAddress)
    {
        $this->emailAddress = $emailAddress;
        return $this;
    }

    public function setFrom($from)
    {
        $this->from = $from;
        return $this;
    }

    public function setSubject($subject)
    {
        $this->subject = $subject;
        return $this;
    }

    public function setBody($body)
    {
        $this->body = $body;
        return $this;
    }

    public function getStatus()
    {
        return $this->status;
    }
```

```
    public function setStatus($status)
    {
        $this->status = $status;
        return $this;
    }

    public function setArguments($arguments)
    {
        $this->arguments = $arguments;
        return $this;
    }

}
```

Done. Doctrine supports Mongo schema managements as well as it does with relational databases. Let's create our first database.

```
$ app/console doctrine:mongodb:schema:create
Created dbs for all classes
Created collections for all classes
Created indexes for all classes
```

Done, doctrine just created for us a database called **app** and an empty **emails** collection.

We will create a new listener that persist email documents to the database.

- Create **src/AppBundle/Event/Listener/EmailCommunicationListener.php**

```
<?php

namespace AppBundle\Event\Listener;

use AppBundle\Communication\Email\Message;
use AppBundle\Document\Email;
use AppBundle\Event\Communication\Email\EmailEvent;
use AppBundle\Event\Communication\Email\EmailSendingEvent;
use Doctrine\Bundle\MongoDBBundle\ManagerRegistry;
use Doctrine\ODM\MongoDB\DocumentManager;

class EmailCommunicationListener
{

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

    public function __construct(ManagerRegistry $registry)
    {
        $this->documentManager = $registry->getManager();
    }
```

```
    public function onBeforeSend(EmailEvent $event)
    {

    }

    public function onEmailSent(EmailSendingEvent $event)
    {
        $this->persistEmailMessage(
                $event->getMessage(),
                $event->getType(),
                Email::STATUS_SENT,
                $event->getArguments()
        );
    }

    public function onTemporaryError(EmailEvent $event)
    {
        $this->persistEmailMessage(
                $event->getMessage(),
                $event->getType(),
                Email::STATUS_TEMPORARY_ERROR,
                $event->getArguments()
        );
    }

    public function onPermanentError(EmailEvent $event)
    {
        $this->persistEmailMessage(
                $event->getMessage(),
                $event->getType(),
                Email::STATUS_PERMANENT_ERROR,
                $event->getArguments()
        );
    }

    private function persistEmailMessage(Message $message,$type,$status,$arguments)
    {
        $email = new Email();
        $email->setType($type);
        $email->setBody($message->getMessage());
        $email->setEmailAddress($message->getTo());
        $email->setFrom($message->getFrom());
        $email->setSubject($message->getSubject());
        $email->setArguments($arguments);
        $email->setStatus($status);

        $this->documentManager->persist($email);
        $this->documentManager->flush();
    }

}
```

- Edit **src/AppBundle/Resources/config/events.xml**, add the following parameters

```
<parameter key="app.email.before_send" type="constant"
>AppBundle\Event\Communication\Email\EmailEvent::BEFORE_SEND</parameter>
<parameter key="app.email.error_temporary" type="constant"
>AppBundle\Event\Communication\Email\EmailEvent::ERROR_TEMPORARY</parameter>
<parameter key="app.email.error_permanent" type="constant"
>AppBundle\Event\Communication\Email\EmailEvent::ERROR_PERMANENT</parameter>
<parameter key="app.email.sent" type="constant"
>AppBundle\Event\Communication\Email\EmailEvent::SENT</parameter>
```

- Edit **src/AppBundle/Resources/config/listeners.yml**, add the following service definition

```
app.email_communication_listener:
    class: AppBundle\Event\Listener\EmailCommunicationListener
    arguments: [@doctrine_mongodb]
    tags:
        - {name: kernel.event_listener,event:
          %app.email.before_send%,method:onBeforeSend}
        - {name: kernel.event_listener,event:
           %app.email.error_temporary%,method:onTemporaryError}
        - {name: kernel.event_listener,event:
           %app.email.error_permanent%,method:onPermanentError}
        - {name: kernel.event_listener,event: %app.email.sent%,method:onEmailSent}
```

- Edit **src/AppBundle/Service/Communication/CommunicationService.php**
  Add `use AppBundle\Document\Email;`
  Update `dispatchEmailSendingEvent` to dispatch the correct event name

```
private function dispatchEmailSendingEvent($type, $arguments, $message, $status)
{
    $event = new EmailSendingEvent($type, $arguments, $message);

    $eventNames = array(
        Email::STATUS_SENT => EmailSendingEvent::SENT,
        Email::STATUS_TEMPORARY_ERROR => EmailSendingEvent::ERROR_TEMPORARY,
        Email::STATUS_PERMANENT_ERROR => EmailSendingEvent::ERROR_PERMANENT
    );
    $this->eventDispatcher->dispatch($eventNames[$status], $event);
}
```

To view the data and manage your MongoDB server, you need a mongo client.
There are plenty of GUIs applications to manage a MongoDB server. I personally prefer to use http://robomongo.org/, is lightwight, cross-platform, and embeds the same JavaScript engine that is used by the the MongoDB mongo shell.
We can already experience the power of MongoDB search. You can try the following query:

```
db.getCollection('emails').find({'arguments.customerName':'contact 1'})
```

This will fetch all the records that has the key **arguments.customerName** equals to **contact 1**

What about `db.getCollection('emails').find({'arguments.customerName':/contact*/})` ? Isn't that awesome? To learn more about querying mongo data, I invite you to visit http://docs.mongodb.org/manual/reference/operator/query/where/

9.4 External provider

We are going to use an external provider to send emails. For this example we will create a fake external provider that just returns on of possible status (success, temporary error, permanent error)

- Create **src/AppBundle/Controller/ExternalProviderController.php**

```
<?php

namespace AppBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\HttpFoundation\JsonResponse;

class ExternalProviderController extends Controller
{

    public function indexAction()
    {
        sleep(rand(0, 5));
        return new JsonResponse(rand(1, 3));
    }

}
```

- Edit **src/AppBundle/Resources/config/routing.yml** and add

```
app_communication:
    resource: "@AppBundle/Resources/config/routing/communication.yml"
    prefix:   /communication
```

- Create **src/AppBundle/Resources/config/routing/communication.yml**

```
communication_external_provider:
    path:     /external_provider
    defaults: { _controller: "AppBundle:ExternalProvider:index" }
```

- Create **src/AppBundle/Communication/Email/ExternalProvider.php**

```php
<?php

namespace AppBundle\Communication\Email;

class ExternalProvider implements ProviderInterface
{

    private $providerHost;

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

    public function send(Message $message)
    {
        $curl = curl_init();
        curl_setopt($curl, CURLOPT_TIMEOUT, 10);
        curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($curl, CURLOPT_URL, $this->providerHost .
                        '/communication/external_provider');
        $result = curl_exec($curl);
        curl_close($curl);
        return $result;
    }

}
```

- Edit **src/AppBundle/Resources/config/services.yml**
  Change **app.email.calls.addProvider** arguments to **@app.provider_email_external**

- Edit **src/AppBundle/Resources/config/email_providers.yml** and add

```
app.provider_email_external:
        class: AppBundle\Communication\Email\ExternalProvider
        arguments: [%communication.external_provider_host%]
```

- Edit **app/config/parameters.yml.dist** and add the parameter
  `communication.external_provider_host: #http://`
  Add the same parameter to **app/config/parameters.yml** the value should be your application's url but with a different host.

On my dev environment I run two instances http://127.0.0.1:8000/ and http://127.0.0.1:8888/ so my parameter looks like `communication.external_provider_host: http://127.0.0.1:8888`

9.5 Email recycler command

We will implement a command that will try to resend any emails that have a temporary error.

- Edit **src/AppBundle/Event/Communication/Email/EmailEvent.php** and add the constant `const RESEND = 'email.resend';`
- Create **src/AppBundle/Command/EmailsRecyclerCommand.php**

```
<?php
namespace AppBundle\Command;

use AppBundle\Communication\Email\Message;
use AppBundle\Document\Email;
use AppBundle\Event\Communication\Email\EmailEvent;
use AppBundle\Service\Communication\CommunicationService;
use Doctrine\ODM\MongoDB\DocumentManager;
use Symfony\Bundle\FrameworkBundle\Command\ContainerAwareCommand;
use Symfony\Component\Console\Input\InputInterface;
use Symfony\Component\Console\Input\InputOption;
use Symfony\Component\Console\Output\OutputInterface;
use Symfony\Component\EventDispatcher\EventDispatcher;

class EmailsRecyclerCommand extends ContainerAwareCommand
{
    /**
     *
     * @var CommunicationService
     */
    private $communicationService;
    /**
     *
     * @var DocumentManager
     */
    private $documentManager;
    /**
     *
     * @var EventDispatcher
     */
    private $eventDispatcher;

    protected function configure()
    {
        parent::configure();
        $this->setName('app:communication:recycler:emails');
        $this->setDescription('Retries the sending of emails with temporary error');
        $this->addOption(
                'limit', '-l',
                InputOption::VALUE_OPTIONAL,
               'Number of emails to resend',
               10
        );
    }
```

```
    protected function initialize(InputInterface $input, OutputInterface $output)
    {
        parent::initialize($input, $output);
        $this->communicationService = $this->getContainer()
                ->get(CommunicationService::ID);
        $this->documentManager = $this->getContainer()
                                      ->get('doctrine_mongodb')
                                      ->getManager();
        $this->eventDispatcher = $this->getContainer()->get('event_dispatcher');
    }

    protected function execute(InputInterface $input, OutputInterface $output)
    {
        $limit = $input->getOption('limit');
        $emailsRepository = $this->documentManager->getRepository(Email::REPOSITORY);
        $emails = $emailsRepository->findBy(
                array('status' => Email::STATUS_TEMPORARY_ERROR), null, $limit
        );

        foreach ($emails as $email) {
            /* @var $email Email */
            $type = $email->getType();
            $emailAddress = $email->getEmailAddress();
            $arguments = $email->getArguments();
            $this->eventDispatcher->dispatch(
                EmailEvent::RESEND, new EmailEvent($type, $emailAddress, $arguments)
            );

            $status = $this->communicationService
                                    ->sendEmail($type, $emailAddress, $arguments);
            $email->setStatus($status);
            $this->documentManager->persist($email);
        }
        $this->documentManager->flush();
    }

}
```

Done, to run the command: `app/console app:communication:recycler:emails`

9.5 Homework

1. When calling the external provider, handle any network or HTTP problems as temporary errors.
2. Update the **Email** document to add **create_date** and **update_date** and **retry_count**