13 Store application

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:symfony-tutorial/app.git
Cloning into 'app'...
remote: Counting objects: 1527, done.
remote: Compressing objects: 100% (685/685), done.
remote: Total 1527 (delta 746), reused 1526 (delta 745), pack-reused 0
Receiving objects: 100% (1527/1527), 1.28 MiB | 1.28 MiB/s, done.
Resolving deltas: 100% (746/746), done.
Checking connectivity... done.
```

Don't forget to `composer install`

Every section's code is in a separate branch with the section's number. So to get the code for the first section of this chapter, just `git checkout 13.1`

13.1 Store application

We will start developing the front website that will interact with the end users. It will be a stand alone application. For this purpose, I created a new git repository https://github.com/symfony-tutorial/store and I added it to packagist https://packagist.org/packages/symfony-tutorial/store

> If you are working with proprietary code, you can to protect it. You can use Github and set your repositories as private (not free), also you can download GitLab
>
> For organizing your project's dependencies you can download packagist and install it locally. You can also use git submodules. I recommend the first option though.

To get the **store application**'s code corresponding to this section, simply run

```
git clone https://github.com/symfony-tutorial/store; cd sore; git checkout 13.1
```

You will need to setup an apache virtual host for the new application. Mine looks like

```
<VirtualHost *:80>
    ServerName store.local
    ServerAlias *.store.local

    DocumentRoot /home/ubuntu/web/store/web
    <Directory /home/ubuntu/web/store/web>
        Options Indexes FollowSymLinks MultiViews
        AllowOverride all
        allow from all
        Require all granted
        DirectoryIndex app.php
        <IfModule mod_rewrite.c>
            RewriteEngine On
            RewriteCond %{REQUEST_FILENAME} !-f [OR]
            RewriteCond %{REQUEST_FILENAME} !=^app_dev.php
            RewriteRule ^(.*)$ /app.php [L]
        </IfModule>
    </Directory>

    ErrorLog /var/log/apache2/store_error.log
    CustomLog /var/log/apache2/store_access.log combined
</VirtualHost>
```

Change the paths accordingly to your environment. The store application was created as a standard symfony application using the command

```
composer create-project 'symfony/framework-standard-edition' store 2.7.3
```

Before starting implementing any logic, we will do the following.

- Add the dependencies `"twig/extensions": "~1.0"`, `"doctrine/mongodb-odm": "1.0.*@dev"` and "doctrine/mongodb-odm-bundle": "3.0.*@dev" to composer.json and `composer update`
- Change the session handler to mongodb.

We will start by implementing basic functionalities.

- Authentication
- Categories listing
- Products listing

Again, we are following a TBD approach, so we will implement the minimum code to get the application running. Afterwards, we will implement each functionality.

- **app/config/routing.yml**

```
store:
    resource: "@AppBundle/Resources/config/routing/store.yml"
    host:     "{country}.store.local"
    prefix:   /

login_route:
    path:     /login
    defaults: { _controller: AppBundle:Security:login }

login_check:
    path: /login_check

logout:
    path: /logout
```

- **src/AppBundle/Controller/StoreController.php**

```php
<?php

namespace AppBundle\Controller;

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

class StoreController extends Controller
{

    public function indexAction(Request $request) {
        $categories = $this->getCategories();
        $products = $this->getProducts();
        return $this->render(
            'AppBundle:Store:index.html.twig',
            array('products' => $products, 'categories' => $categories)
        );
    }

    /**
     *
     * @todo real implementation
     */
    private function getCategories() {
        return array(
            array('label' => 'computers', 'children' => array(
                    array('label' => 'laptops'),
                    array('label' => 'desktop'),
                    array('label' => 'servers')
                )),
            array('label' => 'phones & tablets', 'children' => array(
                    array('label' => 'phones'),
                    array('label' => 'tablets'),
                    array('label' => 'accessories')
                ))
        );
    }

    /**
     *
     * @todo real implementation
     */
    private function getProducts() {
        return array(
        array('id' => 1, 'title' => 'title1', 'description' => 'desc1', 'price' => 1522),
        array('id' => 2, 'title' => 'title2', 'description' => 'desc2', 'price' => 4452),
        array('id' => 3, 'title' => 'title3', 'description' => 'desc3', 'price' => 7755),
        array('id' => 4, 'title' => 'title4', 'description' => 'desc4', 'price' => 4522),
        array('id' => 5, 'title' => 'title5', 'description' => 'desc5', 'price' => 4545),
        );
    }

}
```

- **src/AppBundle/Resources/views/Common/user.html.twig**

```
{% if app.user %}
    {% block loggedin %}
        {{ app.user }}
        <a href="{{url('logout')}}">{{'logout'|trans}}</a>
    {% endblock %}
{% else %}
    {{ render(controller('AppBundle:Security:login'))}}
{% endif %}
```

- **src/AppBundle/Service/CountryService.php**

```
<?php
namespace AppBundle\Service;

use AppBundle\Exception\InvalidCountryCodeException;
use Symfony\Component\HttpKernel\Event\KernelEvent;

class CountryService
{
    const ID = 'app.country';
    const DEFAULT_COUNTRY = 'us';

    private $locale;
    private $countryCode;
    private $currency;

    /**
     *
     * @var \Twig_Environment
     */
    private $twig;

    function __construct(\Twig_Environment $twig) {
        $this->twig = $twig;
    }

    public function onKernelRequest(KernelEvent $event) {
        $countryCode = $event->getRequest()->get('country', self::DEFAULT_COUNTRY);
        $country = $this->getCountryByCode($countryCode);
        $this->currency = $country['currency'];
        $this->locale = $country['locale'];
        $this->countryCode = $countryCode;
        $this->setTwigGlobals();
    }

    public function getCountryByCode($code) {
        $countries = $this->getCountries();
        if (empty($countries[$code])) {
            throw new InvalidCountryCodeException(
              sprintf('Invalid country code %s', $code)
            );
        }
        return $countries[$code];
    }
```

```
    /**
     *
     * @todo real implementation
     */
    public function getCountries() {
        return array(
            'fr' => array('locale' => 'fr', 'currency' => 'EUR'),
            'us' => array('locale' => 'en', 'currency' => 'USD'),
            'ro' => array('locale' => 'ro', 'currency' => 'RON')
        );
    }

    private function setTwigGlobals() {
        $this->twig->addGlobal('country', $this->countryCode);
        $this->twig->addGlobal('currency', $this->currency);
        $this->twig->addGlobal('locale', $this->locale);
    }

    function getLocale() {
        return $this->locale;
    }

    function getCountryCode() {
        return $this->countryCode;
    }

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

}
```

- **src/AppBundle/Resources/config/services.yml**

```
services:
    app.country:
        class: AppBundle\Service\CountryService
        arguments: [@twig]
        tags:
            - { name:kernel.event_listener, event:kernel.request, method:onKernelRequest }
```

13.2 Authentication

First, we will export the user provider as an API from the **app** application.

- Edit **src/AccessBundle/Resources/config/services.yml** and add the tag `{name: json_rpc.service, method: loadUserByUsername}` to `access.user_provider` definition. Alias it to `access`.

Done, now we can get back to the **store** application and implement the authentication.

We add the following parameters to parameters.yml

```
access.url: http://symfony.local/json-rpc/access
access.username: store_api
access.password: pass
```

To communication with the **app** application, we are going to need a JsonRpc client.

- Create **src/AppBundle/Service/JsonRpcClient.php**

```php
<?php

namespace AppBundle\Service;

class JsonRpcClient
{

    private $url;
    private $username;
    private $password;

    function __construct($url, $username, $password) {
        $this->url = $url;
        $this->username = $username;
        $this->password = $password;
    }

    public function call($method, $params = array(), $id = 'id') {
        $data = json_encode(array(
            "jsonrpc" => "2.0",
            "id" => $id,
            "method" => $method,
            "params" => $params,
        ));

        $curl = curl_init($this->url);
        curl_setopt($curl, CURLOPT_CUSTOMREQUEST, "POST");
        curl_setopt($curl, CURLOPT_POSTFIELDS, $data);
        curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($curl, CURLOPT_USERPWD, $this->username . ":" . $this->password);
        curl_setopt($curl, CURLOPT_HTTPHEADER, array(
            'Content-Type: application/json',
            'Content-Length: ' . strlen($data))
        );
        $response = json_decode(curl_exec($curl));
        if (is_string($response)) {
            $response = json_decode($response);
        }

        return $response->result;
    }

}
```

Now we create a custom user provider to retrieve the users from the *app* application.

- Create **src/AppBundle/Security/UserProvider.php**

```php
<?php

namespace AppBundle\Security;

use AppBundle\Service\JsonRpcClient;
use Symfony\Component\Security\Core\Exception\UnsupportedUserException;
use Symfony\Component\Security\Core\User\User;
use Symfony\Component\Security\Core\User\UserInterface;
use Symfony\Component\Security\Core\User\UserProviderInterface;

class UserProvider implements UserProviderInterface
{

    const ID = 'app.user_provider';

    private $jsonRpcClient;

    public function __construct(JsonRpcClient $client) {
        $this->jsonRpcClient = $client;
    }

    public function loadUserByUsername($username) {
        $result = $this->jsonRpcClient->call(
                'loadUserByUsername',
                array('username' => $username)
        );
        $user = $result->result;
        return new User($user->username, $user->password, $user->roles);
    }

    public function refreshUser(UserInterface $user) {
        if (!$user instanceof User) {
            throw new UnsupportedUserException(
            sprintf('Instances of "%s" are not supported.', get_class($user))
            );
        }
        return $this->loadUserByUsername($user->getUsername());
    }

    public function supportsClass($class) {
        $className = '\Symfony\Component\Security\Core\User\User';
        return $class === $className || is_subclass_of($class, $className);
    }

}
```

- Edit **app/config/services.yml** to add the jsonrpc client and the user provider definitions.

```
app.user_provider:
    class: AppBundle\Security\UserProvider
    arguments: ['@jsonrpc_client.access']
    public: false

jsonrpc_client.access:
    class: AppBundle\Service\JsonRpcClient
    arguments: [%access.url%, %access.username%, %access.password%]
    public: false
```

- Edit **app**/**config**/**security.yml** and set the new values for security.encoders and security.providers

```
encoders:
    Symfony\Component\Security\Core\User\User:
        algorithm: sha512
        encode_as_base64: false
        iterations: 0

providers:
    app.user_provider:
        id: app.user_provider
```

13.3 Categories listing

In the store application, add the following parameters

```
catalog.url: http://symfony.local/json-rpc/catalog
catalog.username: api_all
catalog.password: pass
```

- Create **src/AppBundle/Service/CatalogService.php**

```
<?php

namespace AppBundle\Service;

class CatalogService
{

    const ID = 'app.catalog';

    /**
     *
     * @var JsonRpcClient
     */
    private $jsonRpcClient;

    function __construct(JsonRpcClient $jsonRpcClient) {
        $this->jsonRpcClient = $jsonRpcClient;
    }

    public function getCategories() {
        $categories = $this->jsonRpcClient->call('getCategories');
        $parentCategories = array_filter($categories, function ($category) {
            return empty($category->parent_id);
        });

        foreach ($parentCategories as $category) {
          $category->children = array_filter($categories, function($child) use($category){
              return ($child->parent_id === $category->id);
          });
        }

        return $parentCategories;
    }

}
```

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

```
jsonrpc_client.catalog:
  class: AppBundle\Service\JsonRpcClient
  arguments: [%catalog.url%, %catalog.username%, %catalog.password%
  public: false
```

- Edit **src/AppBundle/Resources/config/services.yml** and add the service definition

```
app.catalog:
  class: AppBundle\Service\CatalogService
  arguments: ['@jsonrpc_client.catalog']
```

- Edit **src/AppBundle/Controller/StoreController.php**
  Change `$categories = $this->getCategories();` to `$categories = $this->get(CatalogService::ID)->getCategories();`
  Add `use AppBundle\Service\CatalogService;`
  Remove the method `getCategories`

13.4 Products listing

The app application will publish the products list to elasticsearch. Which will be the products source for the store application.

We will use the elastica bundle to connect to elasticsearch.

13.4.1 App application

Add `"friendsofsymfony/elastica-bundle": "3.1.5"` to composer.json and `composer update`.
Add `new FOS\ElasticaBundle\FOSElasticaBundle(),` to AppKernel.php

- Create **app/config/fos_elastica.yml** and import it in **app/config/config.yml**

```
fos_elastica:
  clients:
      default: { host: %elasticsearch.host%, port: 9200 }
  indexes:
      products:
          index_name: products
          types:
              products:
                  mappings:
                      code: {type: string}
                      title: {type:string}
                      description: {type:string}
                      price: {type:integer}
                      categoryId: {type:integer}
                      category: {type:string}
                      stock: {type:integer}
                  persistence:
                      driver: orm
                      model: AppBundle\Entity\ProductSaleView
                      provider:
                          batch_size: 5000
                      finder: ~
```

Add the parameter `elasticsearch.host: 10.10.10.1`

- Create the following view in the database

```
create view product_sale_view as SELECT
    product.id,
    product.code,
    product.title,
    product.description,
    product_sale.price,
    category.id AS categoryId,
    category.label AS category,
    SUM(product_stock.quantity) AS stock
FROM
    product_sale
        INNER JOIN
    product ON product_sale.product_id = product.id
        LEFT JOIN
    category_has_product ON product.id = category_has_product.product_id
        INNER JOIN
    category ON category.id = category_has_product.category_id
        LEFT JOIN
    product_stock ON product_stock.product_id = product.id
WHERE
    product_sale.active = TRUE
        AND CURRENT_DATE() BETWEEN product_sale.start_date AND product_sale.end_date
GROUP BY product_sale.id
```

- Create the corresponding entity *src/AppBundle/Entity/ProductSaleView.php**

```
<?php

namespace AppBundle\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * ProductSale
 *
 * @ORM\Table(name="product_sale_view")
 * @ORM\Entity(readOnly=true)
 */
class ProductSaleView
{
    const REPOSITORY = 'AppBundle:ProductSaleView';

    /**
     * @var integer
     *
     * @ORM\Column(name="id", type="integer", nullable=false)
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="IDENTITY")
     */
    private $id;
```

```php
    /**
     * @var string
     *
     * @ORM\Column(name="code", type="string", length=45, nullable=true)
     */
    private $code;

    /**
     * @var string
     *
     * @ORM\Column(name="title", type="string", length=200, nullable=true)
     */
    private $title;

    /**
     * @var string
     *
     * @ORM\Column(name="description", type="text", nullable=true)
     */
    private $description;

    /**
     * @var integer
     *
     * @ORM\Column(name="price", type="integer", nullable=true)
     */
    private $price;

    /**
     * @var integer
     *
     * @ORM\Column(name="categoryId", type="integer", nullable=true)
     */
    private $categoryId;

    /**
     * @var string
     *
     * @ORM\Column(name="category", type="integer", nullable=true)
     */
    private $category;

    /**
     * @var integer
     *
     * @ORM\Column(name="stock", type="integer", nullable=true)
     */
    private $stock;

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

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

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

    function getDescription() {
        return $this->description;
    }

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

    function getCategoryId() {
        return $this->categoryId;
    }

    function getCategory() {
        return $this->category;
    }

    function getStock() {
        return $this->stock;
    }

}
```

To publish the products to the elasticsearch index, simply run the command `app/console fos:elastica:populate`

13.4.2 Store application

Add `"friendsofsymfony/elastica-bundle": "3.1.5"` to composer.json and `composer update`.
Add `new FOS\ElasticaBundle\FOSElasticaBundle(),` to AppKernel.php

- Create **app/config/fos_elastica.yml** and import it in **app/config/config.yml**

```
fos_elastica:
  default_manager: mongodb
  clients:
      default: { host: %elasticsearch.host%, port: 9200 }
  indexes:
      products:
          finder: ~
```

Add the parameter `elasticsearch.host: 10.10.10.1`

- Edit **src/AppBundle/Controller/StoreController.php**

```php
<?php

namespace AppBundle\Controller;

use AppBundle\Service\CatalogService;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\HttpFoundation\Request;

class StoreController extends Controller
{

    public function indexAction(Request $request) {
        $term = $request->get('term', '');
        $page = $request->get('page', 1);
        $categories = $this->get(CatalogService::ID)->getCategories();
        $products = $this->getProducts($term, $page);
        return $this->render(
            'AppBundle:Store:index.html.twig',
            array('products' => $products, 'categories' => $categories)
        );
    }

    private function getProducts($term, $page) {
        $from = ($page - 1) * 10;
        $finder = $this->container->get('fos_elastica.client');

        $query = "products/_search?q=*$term*&from=$from&size=10";
        $result = $finder->request($query)->getData()['hits'];

        $products = array_map(function($product) {
            $product['_source']['id'] = $product['_id'];
            return $product['_source'];
        }, $result['hits']);

        return $products;
    }

}
```

13.5 Exercices

We just started implementing a microservices architecture. But there is still work to do.

1. Setup 2 domains to access the application (ie: fr.store.local and us.store.local). If you login to us.store.local then go to fr.store.local you will be asked to login again. Fix it.
2. In the store application, implement a degraded functionality fail-over when elasticsearch is down.
3. In the store application, if no products are found in elasticsearch, fall back to degraded functionality and notify the app application so it populates the products again.
4. Extract the AccessBundle as a stand-alone application.
5. In the store application, implement the following degraded functionality if the access application is not available
    - Use only the user session if the user is already logged in
    - Display a message saying "service unavailable" to users trying to login