10 Security & MongoDb

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 ch9
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`

Make sure MongoDB is running, and comment **elasticsearch** handler from config_dev.yml

10.1 Login form

Before starting, we will update our Symfony and Doctrine versions.

- Edit **composer.json** and update the following packages

```
"symfony/symfony": "2.7.1",
"doctrine/orm": "2.5.0",
"doctrine/dbal": "2.5",
```

Then run `composer:update`

We will start by implementing s simple login Form for our application.

- Edit **app**/**config**/**security.yml** replace it's content by:

```
security:
    providers:
        in_memory:
            memory:
                users:
                    user:
                        password: pass
                        roles: ROLE_USER

    encoders:
        Symfony\Component\Security\Core\User\User: plaintext

    firewalls:
        dev:
            pattern: ^/(_(profiler|wdt|error)|css|images|js)/
            security: false
        login:
            pattern: ^/login
            security: false
        app:
            pattern: .*
            form_login:
                check_path: /check
                login_path: /login
            logout:
                path: /logout
                target: /dashboard
```

- Edit **app/config/routing.yml** and add the following routes:

```
login_route:
    path: /login
    defaults: { _controller: AppBundle:Security:login }
login_check:
    path: /check
logout:
    path: /logout
```

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

```
<?php

namespace AppBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;

class SecurityController extends Controller {

    public function loginAction() {
        $authenticationUtils = $this->get('security.authentication_utils');

        $error = $authenticationUtils->getLastAuthenticationError();
        $lastUsername = $authenticationUtils->getLastUsername();
        if ($error) {
            $this->addFlash('warning', $error->getMessageKey());
        }

        return $this->render(
                        'login.html.twig', array('last_username' => $lastUsername)
        );
    }

}
```

- Create **app/Resources/views/login.html.twig**

```
{% extends 'AppBundle:Dashboard:index.html.twig' %}

{% block items %}
    <div id="div-login" title="Login">
        {% include 'common/flash_messages.html.twig' %}
        <form action="{{ path('login_check') }}" method="post">
            <table width="100%">
                <tr>
                    <td>
                        <label for="username">Username:</label>
                    </td>
                    <td>
                        <input type="text" id="username" name="_username"
                                                value="{{ last_username }}" />
                    </td>
                </tr>
                <tr>
                    <td>
                        <label for="password">Password:</label>
                    </td>
                    <td>
                        <input type="password" id="password" name="_password" />
                    </td>
                </tr>
                <tr>
                    <td>
                    </td>
                    <td>
                        <button type="submit">login</button>
                    </td>
                </tr>
            </table>
        </form>
    </div>
{% endblock %}
```

- Edit **src/AppBundle/Resources/views/Dashboard/index.html.twig**
  wrap the items loop in an *items* block and add `<a href="/logout" id="logout">logout</a>` after **items** block

```
{% block items %}
    {% for item in items %}
        <div class="window"
        {% include 'AppBundle:Dashboard:item_attributes.html.twig'
                  with {item:item} %} >
        </div>
    {% endfor %}
{% endblock %}
<a href="/logout" id="logout">logout</a>
```

- Edit **src/AppBundle/Resources/public/css/dashboard.css** and add

```
#logout{
    float: right;
}
```

- Edit **src/AppBundle/Resources/public/js/dashboard.js** add the following to the end of the load function

```
$('#dashboard .jqdesktop-statusbar').append($('#logout').button());

var loginWindow = $('#div-login').jqWindow({
    width: 300,
    height: 150,
    visible: true,
    showInTaskBar: false,
    showDesktopIcon: false,
    resizable: false,
    maximizable: false,
    closable: false,
    minimizable: false
});

$('#dashboard').jqDesktop('addWindow', loginWindow);
```

Done, we implemented a simple form that matches the login to the configured credentials (user:password).

10.2 Custom user provider

We will create an **AccessBundle** that will manage the user's credentials and roles. We are going to use the generator bundle.

```
$ app/console generate:bundle --bundle-name AccessBundle \
--namespace 'AccessBundle ' --dir src --format yml


  Welcome to the Symfony2 bundle generator



To help you get started faster, the command can generate some
code snippets for you.

Do you want to generate the whole directory structure [no]?


  Summary before generation


You are going to generate a "AccessBundle\AccessBundle" bundle
in "src/" using the "yml" format.

Do you confirm generation [yes]?


  Bundle generation


Generating the bundle code: OK
Checking that the bundle is autoloaded: OK
Confirm automatic update of your Kernel [yes]?
Enabling the bundle inside the Kernel: OK
Confirm automatic update of the Routing [yes]?
Importing the bundle routing resource: OK


  You can now start using the generated code!
```

The custom user provider will load users from MongoDb database. We will create a User document and some data fixtures.

- Create **src/AccessBundle/Document/User.php**

```php
<?php

namespace AccessBundle\Document;

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

/**
 * @MongoDB\Document(db="security", collection="users")
 */
class User
{

    const REPOSITORY = 'AccessBundle:User';

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

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

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

    /**
     * @MongoDB\Collection
     */
    private $roles = array();

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

    public function getUsername() {
        return $this->username;
    }

    public function getPassword() {
        return $this->password;
    }

    public function getRoles() {
        return $this->roles;
    }
```

```
    public function setUsername($username) {
        $this->username = $username;
        return $this;
    }

    public function setPassword($password) {
        $this->password = bin2hex(hash('sha512', $password, true));
        return $this;
    }

    public function setRoles(array $roles) {
        $this->roles = $roles;
        return $this;
    }

    public function addRole($role) {
        $this->roles[] = $role;
    }

}
```

- Create **src/AccessBundle/DataFixtures/MongoDB/UserFixtures.php**

```
<?php

namespace AccessBundle\DataFixtures\MongoDB;

use AccessBundle\Document\User;
use Doctrine\Common\DataFixtures\AbstractFixture;
use Doctrine\Common\Persistence\ObjectManager;

class UserFixtures extends AbstractFixture
{

    private $users = array(
        array('username'=>'user','password' =>'pass',
                  'roles' => array('ROLE_USER')),
        array('username'=>'admin','password' =>'pass',
                  'roles' => array('ROLE_ADMIN')),
        array('username'=>'api_user','password' =>'pass',
                  'roles' => array('ROLE_API')),
    );

    public function load(ObjectManager $manager) {
        foreach ($this->users as $user) {
            $userEntity = new User();
            $userEntity->setUsername($user['username'])
                    ->setPassword($user['password'])
                    ->setRoles($user['roles']);
            $manager->persist($userEntity);
        }
```

```
        for ($i = 1; $i < 100; $i++) {
            $userEntity = new User();
            $userEntity->setUsername("user$i")
                    ->setPassword('pass')
                    ->setRoles(array('ROLE_USER'));
            $manager->persist($userEntity);
        }
        $manager->flush();
    }

}
```

Done, we can create the database and load the users fixtures.

```
$ app/console doctrine:mongodb:fixtures:load
  > purging database
  > loading AccessBundle\DataFixtures\MongoDB\UserFixtures
```

To create our custom user provider we will need a service that implements `Symfony\Component\Security\Core\User\UserProviderInterface` and a User class that implements `Symfony\Component\Security\Core\User\UserInterface`

- Create **src/AccessBundle/User.php**

```
<?php

namespace AccessBundle;

use Symfony\Component\Security\Core\User\EquatableInterface;
use Symfony\Component\Security\Core\User\UserInterface;

class User implements UserInterface, EquatableInterface
{

    private $username;
    private $password;
    private $salt;
    private $roles;

    public function __construct(
        $username, $password, array $roles = array(), $salt = null
    ) {
        $this->username = $username;
        $this->password = $password;
        $this->salt = $salt;
        $this->roles = $roles;
    }

    function getUsername() {
        return $this->username;
    }
```

```php
    function getPassword() {
        return $this->password;
    }

    function getSalt() {
        return $this->salt;
    }

    function getRoles() {
        return $this->roles;
    }

    function setUsername($username) {
        $this->username = $username;
    }

    function setPassword($password) {
        $this->password = $password;
    }

    function setSalt($salt) {
        $this->salt = $salt;
    }

    function setRoles($roles) {
        $this->roles = $roles;
    }

    public function eraseCredentials() {

    }

    public function isEqualTo(UserInterface $user) {
        if (!$user instanceof User) {
            return false;
        }

        if ($this->password !== $user->getPassword()) {
            return false;
        }

        if ($this->salt !== $user->getSalt()) {
            return false;
        }

        if ($this->username !== $user->getUsername()) {
            return false;
        }
        return true;
    }
}
```

- Create **src/AccessBundle/UserProvider.php**

```php
<?php

namespace AccessBundle;

use AccessBundle\Document\User as UserDocument;
use Doctrine\Bundle\MongoDBBundle\ManagerRegistry;
use Doctrine\ODM\MongoDB\DocumentManager;
use Doctrine\ODM\MongoDB\DocumentRepository;
use Symfony\Component\Security\Core\Exception\UnsupportedUserException;
use Symfony\Component\Security\Core\Exception\UsernameNotFoundException;
use Symfony\Component\Security\Core\User\UserInterface;
use Symfony\Component\Security\Core\User\UserProviderInterface;

class UserProvider implements UserProviderInterface
{

    const ID = 'access.user_provider';

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

    /**
     *
     * @var DocumentRepository
     */
    private $repository;

    public function __construct(ManagerRegistry $registry) {
        $this->manager = $registry->getManager();
        $this->repository = $this->manager
                                ->getRepository(UserDocument::REPOSITORY);
    }


    public function loadUserByUsername($username) {
        /* @var $user UserDocument  */
        $user = $this->repository->findOneByUsername($username);
        if (!$user) {
            throw new UsernameNotFoundException();
        }
        return new User(
                $user->getUsername(), $user->getPassword(), $user->getRoles()
        );
    }
```

```
    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 = '\AccessBundle\UserProvider';
        return $class === $className || is_subclass_of($class, $className);
    }

}
```

We need to register UserProvider as a service

- Edit **src/AccessBundle/Resources/config/services.yml**

```
services:
    access.user_provider:
        class: AccessBundle\UserProvider
        arguments: [@doctrine_mongodb]
```

Now we have to tell the security component to use our service as a user provider.

- Edit **app/config/security.yml**

Remove **in_memory** provider and add the following provider

```
access_service:
    id: access.user_provider
```

Remove **Symfony\Component\Security\Core\User\User: plaintext** encoder and add the following encoder

```
AccessBundle\User:
    algorithm: sha512
    encode_as_base64: false
    iterations: 0
```

10.3 User roles

So far, we just implemented an authentication mechanism, that means *who can access the application*. Now we will add the ability to add authorization which will decide *who can access a specific part of the application*

We will define the following roles:

- *Api* can access the json-rpc services
- *Admin* can access the admin area
- *User* can access the rest of the application

For the API we will use basic HTTP authentication.

- Edit **app**/**config**/**security.yml** and add the following firewall **before** app

```
api:
    pattern: ^/json-rpc
    http_basic: ~
```

In the same file, add the following access control rules as first level under **sercurity**

```
access_control:
    - { path: ^/json-rpc, role: ROLE_API}
    - { path: ^/admin, role: ROLE_ADMIN}
    - { path: ^/, role: [ROLE_USER, ROLE_ADMIN]}
```

10.4 Administration interface

So far, we created users using data fixtures. We would like to have an administration interface that:

- Create users
- Can be accessible only by users having the *ADMIN* role

We are going to use jqGrid, let's start by downloading it.

```
$ wget https://github.com/tonytomov/jqGrid/archive/v4.8.2.zip
$ unzip v4.8.2.zip -d  src/AppBundle/Resources/public/
```

We will create a UserService that will fetch and persist users.

- Create **src/AccessBundle/UserService.php**

```php
<?php

namespace AccessBundle;

use Doctrine\Bundle\MongoDBBundle\ManagerRegistry;
use Doctrine\ODM\MongoDB\DocumentManager;
use Doctrine\ODM\MongoDB\DocumentRepository;

class UserService
{

    const ID = 'access.user';

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

    /**
     *
     * @var DocumentRepository
     */
    private $repository;

    public function __construct(ManagerRegistry $registry) {
        $this->manager = $registry->getManager();
        $this->repository = $this->manager
                                ->getRepository(Document\User::REPOSITORY);
    }

    public function find($username, $start = 0, $limit = 10) {
        $queryBuilder = $this->manager
                             ->createQueryBuilder(Document\User::REPOSITORY);
        $queryBuilder->limit($limit)->skip($start);

        if ($username) {
            $queryBuilder->field("username") . equals($username);
        }

        return $queryBuilder->getQuery()->execute();
    }


    public function count() {
        $queryBuilder = $this->manager
                             ->createQueryBuilder(Document\User::REPOSITORY);
        return $queryBuilder->getQuery()->count();
    }
```

```
    public function create($username, $password, $roles) {
        $user = new Document\User();
        $user->setUsername($username)
                ->setPassword($password)
                ->setRoles($roles);
        $this->manager->persist($user);
        $this->manager->flush();
    }

}
```

And we need to register the service.

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

```
access.user:
    class: AccessBundle\UserService
    arguments: [@doctrine_mongodb]
```

We will create some views and javascript to handle the users grid.

- Create **src/AppBundle/Resources/views/Admin/index.html.twig**

```
{% extends 'common/app.html.twig' %}

{% block javascripts %}
  {{ parent() }}
  <script type="text/javascript"
  src="{{asset('bundles/app/jqGrid-4.8.2/js/minified/i18n/grid.locale-en.js')}}" >
  </script>
  <script type="text/javascript"
  src="{{asset('bundles/app/jqGrid-4.8.2/js/minified/jquery.jqGrid.min.js')}}" >
  </script>
  <script type="text/javascript"
          src="{{asset('bundles/app/js/admin.js')}}" >
  </script>
{% endblock %}


{% block stylesheets %}
    {{parent()}}
    <link rel="stylesheet"
          href="{{asset('bundles/app/jqGrid-4.8.2/css/ui.jqgrid.css')}}" >
{% endblock %}


{% block header%}
    <h1>{{'administration'|trans|capitalize}}</h1>
{% endblock %}
```

```
{% block content%}
    <div id="admin-tabs">
        <ul>
            <li><a href="{{ path('admin_users')}}">Users</a></li>
        </ul>
    </div>
{% endblock %}
```

- Create **src/AppBundle/Resources/views/Admin/users.html.twig**

```
<table id="grid-users" width="100%"></table>
<div id="pager-users"></div>

<script type="text/javascript">
    adminGrid();
</script>
```

- Create **src/AppBundle/Resources/public/js/admin.js**

```
$(function () {
    $('#admin-tabs').tabs();
});


function adminGrid() {
    $("#grid-users").jqGrid({
        url: '/admin/users/find',
        colNames: ['Username', 'Roles', 'Password'],
        colModel: [
            {name: 'username', index: 'username', editable: true},
            {'name': 'roles', index: 'roles', editable: true},
            {'name': 'password', index: 'password', editable: true}
        ],
        pager: '#pager-users',
        datatype: "json",
        viewrecords: true,
        editurl: '/admin/users/edit',
        autowidth: true,
        height: 'auto'
    });
    $("#grid-users").jqGrid('navGrid',
            '#pager-users',
            {edit: true, add: true, del: true}
    );
    $("#grid-users").jqGrid('inlineNav', "#grid-users");
}
```

We need a controller to handle the actions.

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

```php
<?php

namespace AppBundle\Controller;

use AccessBundle\UserService;
use stdClass;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\HttpFoundation\JsonResponse;
use Symfony\Component\HttpFoundation\Request;

class AdminController extends Controller
{

    public function indexAction() {
        return $this->render('AppBundle:Admin:index.html.twig');
    }

    public function listUsersAction() {

        return $this->render('AppBundle:Admin:users.html.twig');
    }

    public function findUsersAction(Request $request) {
        $username = $request->get('_search');
        $page = $request->get('page');
        $rows = $request->get('rows');

        if ($username === 'false') {
            $username = false;
        }
        /* @var $userServie UserService */
        $userService = $this->get(UserService::ID);
        $total = $userService->count();
        $totalPages = ceil($total / $rows);

        $responce = new stdClass();
        $responce->page = $page;
        $responce->total = $totalPages;
        $responce->records = $total;
        $users = $userService->find($username, ($page - 1) * $rows, $rows);
        foreach ($users as $user) {
            $row = array($user->getUsername(), $user->getRoles());
            $responce->rows[] = array('id' => $user->getId(), 'cell' => $row);
        }

        return new JsonResponse($responce);
    }
```

```
    public function editUserAction(Request $request) {
        /* @var $userServie UserService */
        $userService = $this->get(UserService::ID);

        $action = $request->get('oper');
        $username = $request->get('username');
        $password = $request->get('password');
        $roles = explode(',', $request->get('roles'));
        if ($action === 'add') {
            $userService->create($username, $password, $roles);
        }

        return new JsonResponse();
    }

}
```

We should now update the routing configuration.

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

```
admin:
    path:     /
    defaults: { _controller: "AppBundle:Admin:index" }
admin_users:
    path:     /users
    defaults: { _controller: "AppBundle:Admin:listUsers" }
admin_users_find:
    path:     /users/find
    defaults: { _controller: "AppBundle:Admin:findUsers" }
admin_users_edit:
    path:     /users/edit
    defaults: { _controller: "AppBundle:Admin:editUser" }
```

- Edit **src/AppBundle/Resources/config/routing.yml** and add the following route definition

```
app_admin:
    resource: "@AppBundle/Resources/config/routing/admin.yml"
    prefix:   /admin
```

We are almost done, we just need to add a link to the admin interface from the dashboard.

- Edit **app/config/dashboard.yml** and add to the items

```
admin:
    route: admin
```

10.5 Authorization voter

We already did a good job by securing our API. We want to have more control over the clients access. We would like to give access not just to the API, but to a specific service.

So, what we want to achieve is that a user can call a specific service through the API only if he has the role corresponding to that service.

Example: to access the **order** service from our API, the user must have at least two roles, ROLE_API and ROLE_API_ORDER

Let's update our users data fixtures to load some authorized users into the database.

- Edit **src**/**AccessBundle**/**DataFixtures**/**MongoDB**/**UserFixtures.php** and add the following users:

```
array('username' => 'api_warehouse', 'password' => 'pass',
            'roles' => array('ROLE_API', 'ROLE_API_WAREHOUSE')),
        array('username' => 'api_order', 'password' => 'pass',
            'roles' => array('ROLE_API', 'ROLE_API_ORDER')),
        array('username' => 'api_all', 'password' => 'pass',
            'roles' => array('ROLE_API', 'ROLE_API_ORDER', 'ROLE_API_WAREHOUSE')),
```

To achieve this functionality, we will create am authorization voter.

- Create **src/AccessBundle/ApiVoter.php**

```php
<?php

namespace AccessBundle;

use Symfony\Component\Security\Core\Authorization\Voter\AbstractVoter;
use Symfony\Component\Security\Core\User\UserInterface;

class ApiVoter extends AbstractVoter
{

    /**
     *
     * @var UserService
     */
    private $userService;

    protected function getSupportedAttributes() {
        return array();
    }

    public function supportsAttribute($attribute)
    {
        return true;
    }

    protected function getSupportedClasses() {
        return array('JsonRpcBundle\Server');
    }

    protected function isGranted($attribute, $object, $user = null) {
        if (!$user instanceof UserInterface) {
            return false;
        }
        $role = sprintf('ROLE_API_%s', strtoupper($attribute));
        return in_array($role, $user->getRoles());
    }

}
```

We need to register our voter as a service and tag it as **security.voter**

- Edit **src/AccessBundle/Resources/config/services.yml** and add the service defintion

```
access.api_voter:
        class:      AccessBundle\ApiVoter
        public:     false
        tags:
           - { name: security.voter }
```

All we need to do now, is to update the JsonRpc controller to check for access.

- Edit **src/JsonRpcBundle/Controller/ServerController.php**
  After `$server = $this->get(Server::ID);` add:

```
$authChecker = $this->get('security.authorization_checker');
if (false === $authChecker->isGranted($service, $server)) {
    throw $this->createAccessDeniedException('Access denied');
}
```

To allow the access to our fake externale provider, edit **app/config/security.yml** and add the following firewall after **api**

```
external_provider:
    pattern: ^/communication/external_provider
    security: false
```

Done, try to experiment calling the API with various users and see the result.

10.6 Homework

1. Create and OrderVoter that will restrict editing or creating orders only to users having the role ROLE_API_ORDER
2. Besides the Basic access authentication, add the possibility to accept Digest access authentication for the json-rpc API