

[pawel.rajba@gmail.com](mailto:pawel.rajba@gmail.com), <http://kursy24.eu/>

# Symfony2

---

# Agenda

- Wprowadzenie, struktura
- Podstawy Twig
- Adres i nawigacja
- Request, Response i parametry
- Ciasta, sesje i flash messages
- Model i baza danych
- Formularze
- Security

# Wprowadzenie

- Framework za free o ogromnej społeczności
- Obecnie dostępny w wersji 2 – nowa jakość
  - Oparty o PHP 5.3 (wersja minimalna)
- Wsparcie ze strony NetBeans i Eclipse
- Integracja z Doctrine i Twig
- Wsparcie w wielu obszarach
  - np. świetne formularze z walidacją, uwierzytelnianie i autoryzacja, dostęp do danych, debug toolbar
- Wykorzystywany przez wiele dużych serwisów
  - np. Yahoo

# Wprowadzenie

- Do pobrania ze strony [symfony.com](http://symfony.com)
  - wersje z vendors i bez vendors
- Struktura jak po prawej:
- Bundle
  - plik `app\AppKernel.php`
- Adnotacje
- Konsola:
  - `php app/console`

```
www/ <- your web root directory
Symfony/ <- the unpacked archive
  app/
    cache/
    config/
    logs/
    Resources/
  bin/
  src/
    Acme/
      DemoBundle/
        Controller/
        Resources/
        ...
  vendor/
    symfony/
    doctrine/
    ...
  web/
    app.php
    ...
```

# Przykład

- review
  - Uruchamiamy aplikację
    - [http://localhost/sf/review/web/app\\_dev.php/](http://localhost/sf/review/web/app_dev.php/)
    - <http://localhost/sf/review/web/app.php/>
  - Przechodzimy konfigurację (link **configure**)
  - Uruchamiamy demo (Hello World)
    - [http://localhost/sf/web/app\\_dev.php/demo/hello/Pawel](http://localhost/sf/web/app_dev.php/demo/hello/Pawel)
    - Patrzymy gdzie są kontrolery i widoki
    - Omówienie jak to z grubsza działa
  - Oglądamy konsolę

# Struktura

- Główne składowe
  - Kontrolery
    - folder `src\<BUNDLE>\Controller`
  - Widoki (oparte o Twig)
    - folder `src\<BUNDLE>\Resources\views`

# Struktura

- Powiązanie kontrolera z widokiem:

- Podejście natywne:

```
public function indexAction() {  
    $response = $this->  
        render('SiteBundle:Default:index.txt.twig');  
    $response->headers->  
        set('Content-Type', 'text/plain');  
    return $response;  
}
```

- Wykorzystanie adnotacji

```
/**  
 * @Route("/hello/{name}", name="_demo_hello")  
 * @Template()  
 */  
public function helloAction($name) {  
    return array('name' => $name);  
}
```

# Podstawy Twig

- Wyświetlenie zmiennej:
  - `{{ z }} {{ z.x }} {{ t['key'] }} {{ p.getName() }}`
- Bloki:
  - `{% for i in tab %}<li>{{ i }}</li>{% endfor %}`
  - `{% if count > 10 %}{{ pager }}{% endif %}`
- Wstawianie „assets”
  - `<link href="{{ asset('css/blog.css') }}" rel="stylesheet" type="text/css" />`
  - ``
- Wstawianie innego szablonu
  - `{% include 'SiteBundle:Helper:pager.html.twig', with { 'parameters' : { 'count': 10 } } %}`
- Strona domowa: <http://twig.sensiolabs.org/>
  - Twig będzie omawiany na kolejnych wykładach



# Podstawy Twig

- Korzystając z możliwości Twig, budujemy strukturę opartą o szablony

- Plik widoku

```
{# src/SiteBundle/Resources/views/Demo/hello.html.twig #}  
{% extends "SiteBundle::layout.html.twig" %}  
{% block content %}<h1>Hello {{ name }}!</h1>{% endblock %}
```

- Plik szablonu (layoutu)

```
{# src/SiteBundle/Resources/views/layout.html.twig #}  
<!DOCTYPE html>  
<html>  
  (...)   
<body>  
  (...)<div class="content">{% block content %}{% endblock %}</div>(...)   
</body>  
</html>
```

# Adresy i nawigacja

- Routing (mapowanie adresów) określamy
  - W pliku `app/config/routing[_dev].yml`, przykład
    - `_default:`
      - pattern: /
      - defaults: { \_controller: SiteBundle:Default:index }
    - Poprzez adnotacje, przykład
      - `@Route("/", name="_demo")`
      - a w pliku `routing.yml` skierowanie na kontroler(y)
        - SiteBundle:
        - resource: "@SiteBundle/Controller/"
        - type: annotation
        - prefix: /
    - Większy przykład
      - `@Route("/zgloszenia/{page}", name="pages_zgloszenia", requirements = {"page" = "\d+"}, defaults = {"page" = 1})`

# Adresy i nawigacja

## ■ Po stronie kontrolera mamy

- ```
return $this->redirect(  
    $this->generateUrl('_demo_hello',  
        array('name' => 'Lucas')));
```
- ```
$response = $this->forward(  
    'AcmeDemoBundle:Hello:fancy',  
    array('name' => $name, 'color' => 'green'));
```

## ■ Po stronie szablonu

- ```
<a href="  
    {{ path('_demo_hello', { 'name': 'Thomas' }) }}  
">Greet Thomas!</a>
```
- ```
<a href="  
    {{ url('blog_show', { 'slug': 'my-blog-post' }) }}  
">previous post</a>
```

# Przykład

---

- layoutnavigation

# Request, Response i parametry

- Request – obiekt żądania
  - Dostęp: `$this->getRequest()`
  - `$request->isXmlHttpRequest();`
    - Czy AJAX?
  - `$request->get('page');`
    - Parametr (get lub post)
- Response – obiekt odpowiedzi
  - Oglądamy metody w przykładzie
- Przegląd składowych podczas demonstracji

# Request, Response i parametry

- Przekazywanie parametrów:

- Definicja:

- `# app/config/routing.yml`  
hello:  
    pattern: /hello/{first\_name}/{last\_name}  
    defaults: { \_controller: AppBundle :Hello:index, color: green }

- Akcja

- `public function indexAction($first_name, $last_name, $color)`  
    {  
        // ...  
    }

# Request, Response i parametry

- Przekazywanie parametrów, uwagi:
  - Kolejność parametrów nie ma znaczenia
    - Rozpoznawanie po nazwie
  - Metoda akcji nie musi mieć wszystkich parametrów
    - Ale wszystkie przekazane parametry muszą być ujęte w definicji routingu,...
    - chociaż można dodawać dowolną liczbę parametrów domyślnych

# Request, Response i parametry

- Przekazywanie parametrów
  - Albo po prostu:
    - ```
public function showAction(Request $r)  
{  
}
```
    - Tylko trzeba pamiętać, aby dodać:  

```
use Symfony\Component\HttpFoundation\Request;
```



# Przykład

---

- request

# Ciastka

- Pobranie

- `$ciacho = $this->getRequest()  
->cookies->getInt("ciacho", 0);`

- Zapisanie

- Tradycyjnie

- `setcookie("ciacho", $ciacho+1);`

- Korzystając z obiektu response

- `$response = new Response();  
$response->headers->setCookie(new Cookie('ciacho', 100));  
$response->setContent($this->render(  
 'SimpleSiteBundle:Default:cookie.html.twig', $params));  
return $response;`

# Sesje i flash messages

- Obsługa sesji
  - Pobranie
    - `$session = $this->getRequest()->getSession();`
  - Zapisanie zmiennej
    - `$session->set('foo', 'bar');`
  - Pobranie
    - `$foo = $session->get('foo');`
- Można także łatwo przechowywać sesje w bazie danych
  - Zobaczymy na przykładzie, jak to zrobić

# Sesje i flash messages

- Flash messages
  - Krótkie wiadomości tekstowe
  - Widoczność do następnego żądania
  - Zastosowanie: powiadomienia typu „Dane zapisano”, „Błędny formularz” ...
  - Przykład:
    - ```
public function updateAction() {  
    $form = $this->createForm(...);  
    $form->bindRequest($this->getRequest());  
    if ($form->isValid()) {  
        // do some sort of processing  
        $this->get('session')->setFlash('notice', 'Your changes were saved!');  
        return $this->redirect($this->generateUrl(...));  
    }  
    return $this->render(...);  
}
```
    - ```
{% if app.session.hasFlash('notice') %}  
    <div class="flash-notice">{{ app.session.flash('notice') }}</div>  
{% endif %}
```

# Przykład

- cookiesession
  - Przeglądamy DefaultController
  - Oglądamy działanie flashMessages i sesji
  - Przechodzimy do SessionController
  - Na koniec oglądamy config.yml
- Więcej:
  - [http://symfony.com/doc/2.0/cookbook/configuration/pdo\\_session\\_storage.html](http://symfony.com/doc/2.0/cookbook/configuration/pdo_session_storage.html)

# Model i baza danych

- Dostęp do bazy oparty na PDO i Doctrine
- Generatory modelu, CRUD i formularzy
- ...czyli łatwo lekko i przyjemnie

# Model i baza danych

- Kolejne kroki
  - Tworzymy bazę danych z tabelami
  - Tworzymy klasy
    - `.\php app/console doctrine:mapping:import SimpleSiteBundle annotation`
  - Tworzymy settery i gettery
    - `.\php app/console doctrine:generate:entities SimpleSiteBundle`
      - Niestety ręcznie dopisujemy relacje
  - Możemy dodatkowo utworzyć prostą obsługę CRUD:
    - `.\php app/console doctrine:generate:crud --entity="SimpleSiteBundle:Osoba" ---route-prefix="/osoba" --with-write --format="annotation,,`

# Przykład

---

- database



# Model i baza danych

- Podstawy
  - <http://symfony.com/doc/current/book/doctrine.html>
- Tworzenie encji na podstawie bazy danych
  - [http://symfony.com/doc/2.0/cookbook/doctrine/reverse\\_engineering.html](http://symfony.com/doc/2.0/cookbook/doctrine/reverse_engineering.html)

# Formularze

- Pełna integracja z Twig
- Bardzo przejrzyste i eleganckie rozwiązanie
- Możliwość definicji formularzy
  - Ad-hoc w kontrolerze
  - Elegancko w osobnej klasie
- Definicja walidacji
  - W encjach
  - W klasie formularza niezależnie od modelu
- Mechanizm Form Theming

# Formularze

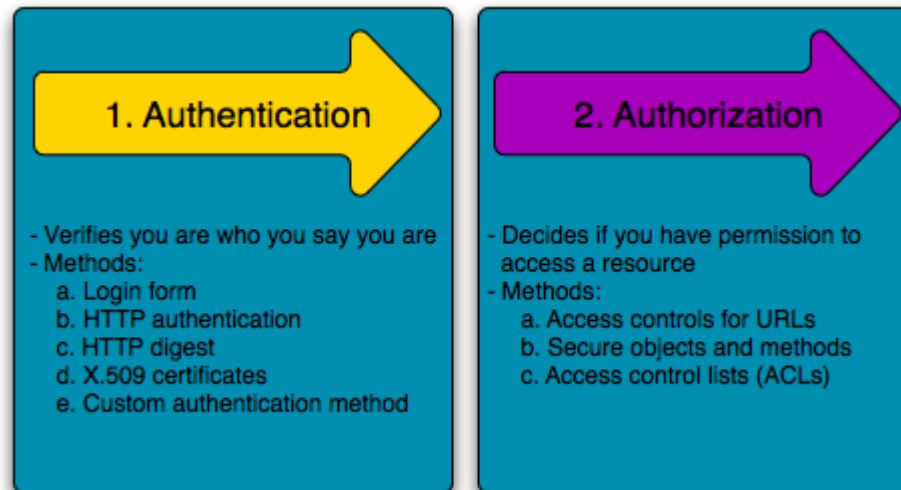
- Przegląd kontrolek
  - <http://symfony.com/doc/2.0/reference/forms/types.html>
- Przegląd walidatorów
  - <http://symfony.com/doc/2.0/reference/constraints.html>
- Wprowadzenie do formularzy
  - <http://symfony.com/doc/2.0/book/forms.html>

# Przykład

- forms
  - form1
  - form2
  - form3
  - form4
  - form5

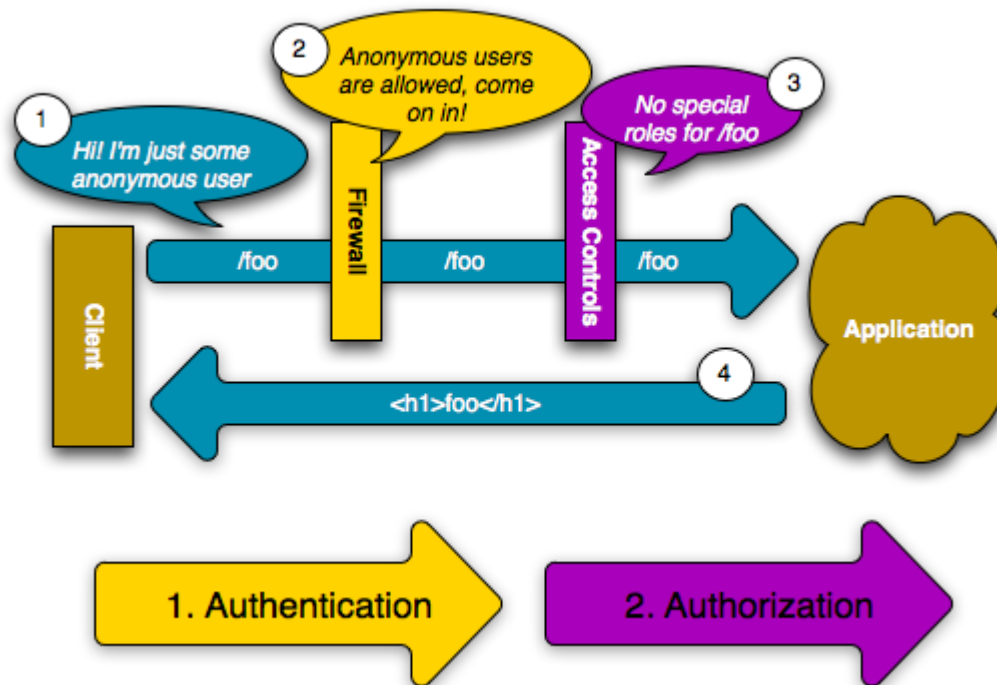
# Security

- Security to podsystem Symfony2 umożliwiające sterowanie dostępem
- Konfiguracja w `app/config/security.yml`
- Standardowo mamy dwa etapy



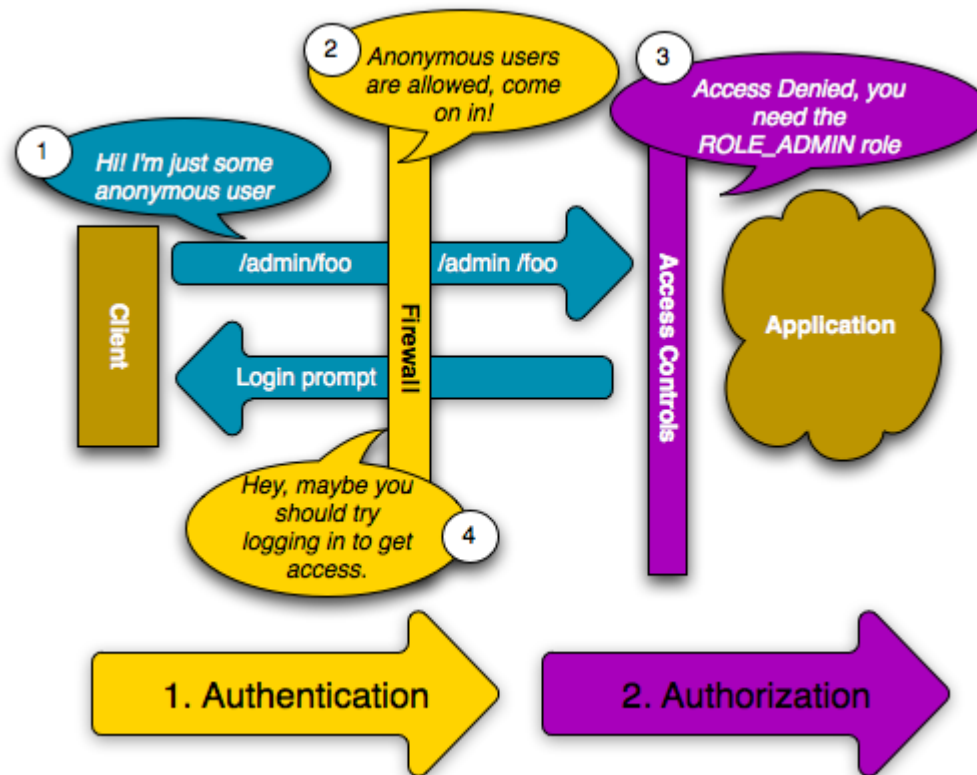
# Security

- Przykładowe scenariusze sterowania dostępem:  
(1)



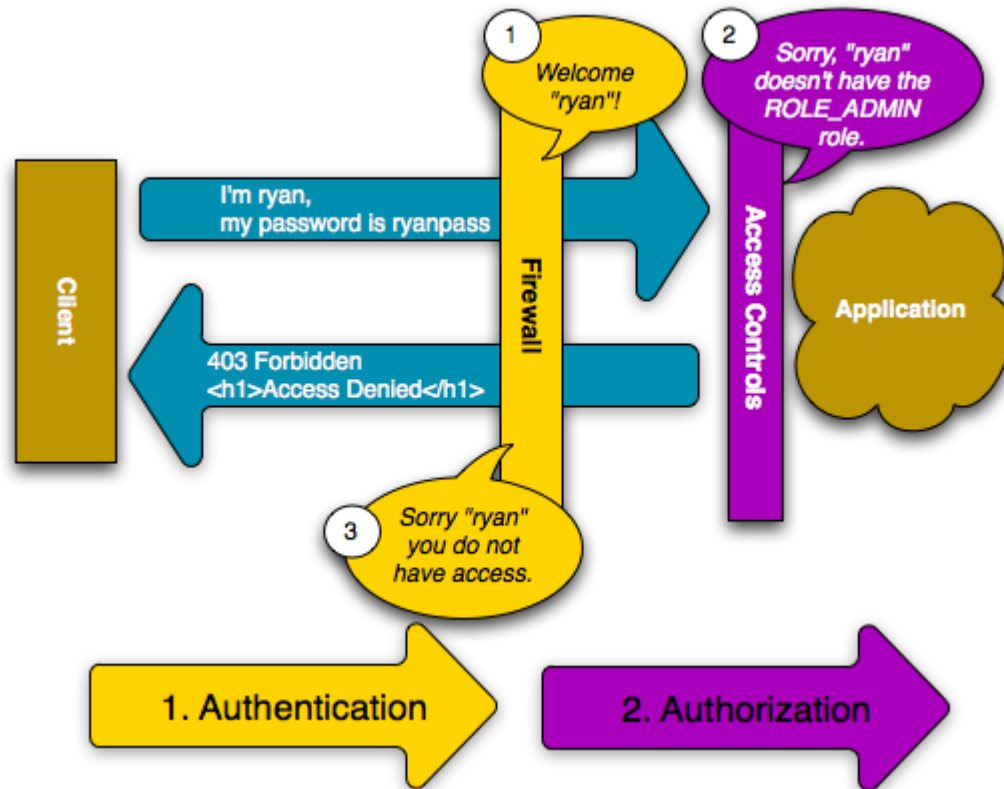
# Security

- Przykładowe scenariusze sterowania dostępem:  
(2)



# Security

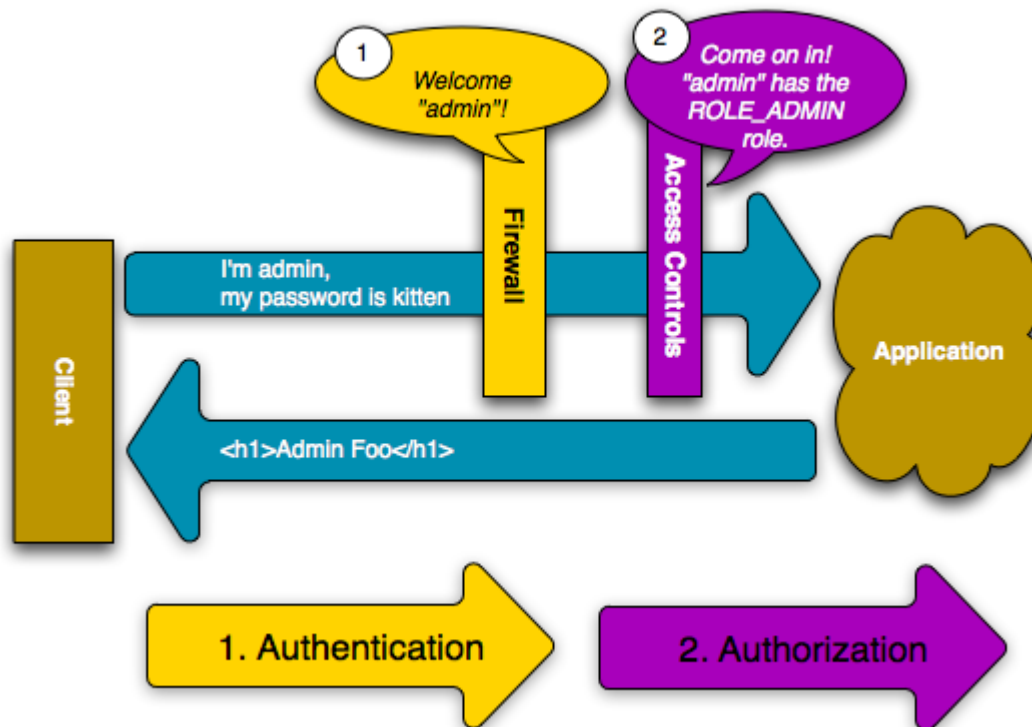
- Przykładowe scenariusze sterowania dostępem:  
(3)





# Security

- Przykładowe scenariusze sterowania dostępem:  
(4)



# Security

- Podstawy zestaw, aby uruchomić „security”
  - Definicja firewalla, czyli przede wszystkim
    - Co jest zabezpieczone (wzorzec ścieżki, np. ^/)
    - Mechanizm uwierzytelniania (np. form\_login)
  - Określenie dostawcy kont użytkowników
  - Określenie sposobu kodowania haseł (np. plaintext)
  - Określenie autoryzacji
    - Do których ścieżek (np. ^/admin)
    - Które role (np. ROLE\_ADMIN)

# Security

- Prosty przykład:

```
# app/config/security.yml
security:
  firewalls:
    secured_area:
      pattern:    ^/
      anonymous:  ~
      http_basic:
        realm: "Secured Demo Area"

  access_control:
    - { path: ^/admin, roles: ROLE_ADMIN }

  providers:
    in_memory:
      users:
        ryan: { password: ryanpass, roles: 'ROLE_USER' }
        admin: { password: kitten, roles: 'ROLE_ADMIN' }

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

# Security

- Uwierzytelnianie przez formularz HTML
  - Tworzymy dwie ścieżki
    - login – formularz logowania (robimy sami)
    - login\_check – weryfikacja tożsamości (automat)
  - Do security.yml dopisujemy
    - form\_login:
      - check\_path: /login\_check
      - login\_path: /login
  - Tworzymy odpowiedni widok formularza i akcję kontrolera

# Security

- Autoryzacja i role
  - Dostęp jest określany na podstawie ról
  - Definicje ról są w sekcji `role_hierarchy`
    - Możemy mieć dziedziczenie ról
  - Sterowanie dostępem jest w `security.yml` w sekcji `access_control`
    - Lista ze ścieżkami i wymaganymi rolami
    - Uwaga: brana jest pierwsza reguła, która się dopasuje

# Security

- Sprawdzanie ról można zrobić również w kontrolerze:

```
public function indexAction()  
{  
    // show different content to admin users  
    if ($this->get('security.context')->isGranted('ADMIN')) {  
        // Load admin content here  
    }  
    // load other regular content here  
}
```

# Przykład

---

- security

# Security

- Szyfrowanie haseł
  - Mamy do dyspozycji przede wszystkim funkcje skrótu, np. sha1, sha512 (to jest wbudowane)
    - Pełna lista jest zwracana przez funkcję `hash_algos()`
    - Implementacja w klasie `vendor\symfony\src\Symfony\Component\Security\Core\Encoder\MessageDigestPasswordEncoder.php`
  - Można także zdefiniować szyfrowanie dowolne (np. symetryczne)
    - Ale to trzeba zrobić ręcznie przez definicję usługi



# Security

- Użytkownicy w bazie danych
  - Tworzymy tabelkę w bazie danych
  - Definiujemy encję opisującą użytkownika
    - Jest to klasa implementująca  
Symfony\Component\Security\Core\User\UserInterface
  - W security.yml dopisujemy  
providers:  
main:  
entity: { class: Simple\SiteBundle\Entity\User, property: username }

# Przykład

---

- security2