Programowanie Aplikacji Internetowych

Laboratorium nr 4

Budowa aplikacji opartej o RESTową architekturę.

Poniżej znajdują się zadania, które należy wykonać w ramach laboratoriów, a następnie sporządzić sprawozdanie w formie archiwum .zip. Plik archiwum powinien mieć nazwę zgodną ze wzorem: PAI_Lab<nr_laboratorium>_<pierwsza_litera_imienia>.<nazwisko_bez_polskich_znaków>.zip, np. PAI_Lab2_J.Kowalski.zip. W archiwum powinny znajdować się wszystkie pliki z poniższych zadań.

Zadanie 1 Instalacja Symfony 2

Na wstępie należy zainstalować PHP w najnowszej wersji oraz serwer bazy danych MySQL. Po instalacji sprawdzamy, czy wszystko działa poprawnie wpisując w konsoli:

```
php –version
```

Następnie instalujemy symfonię na **Linuxie**:

```
php -r "readfile('http://symfony.com/installer');" > symfony.phar
sudo mv symfony.phar /usr/local/bin/symfony
chmod a+x /usr/local/bin/symfony
symfony
```

Na **Windowsie**:

```
php -r "readfile('http://symfony.com/installer');" > symfony.phar
php symfony.phar
```

Zadanie 2 Instalacja Composer

Wpisujemy w konsoli:

```
php -r "readfile('https://getcomposer.org/installer');" | php
```

Zadanie 3 RESTowy serwer w Symfony 2

Tworzymy nowy projekt na Linuxie:

```
symfony new myrest
```

Na **Windowsie**:

```
php symfony.phar new myrest
```

Uruchamiamy aplikację:

```
cd myrest/
php app/console server:run
```

W przeglądarce wpisujemy adres http://localhost:8000/.

Instalujemy nasz ulubiony FOSRestBundle oraz dodatkowy bundle, który przyda nam się do obsługi serializacji obiektów:

```
php ../composer.phar require friendsofsymfony/rest-bundle
php ../composer.phar require jms/serializer-bundle
```

Dodajemy bundle do Symfony w pliku *myrest/app/AppKernel.php*:

```
<?php
...
class AppKernel extends Kernel
{
    public function registerBundles()
    {
        $bundles = array(
            ...
```

```
            new AppBundle\AppBundle(),
            new FOS\RestBundle\FOSRestBundle(),
            new JMS\SerializerBundle\JMSSerializerBundle(),
        );
        ...
    }
    ...
}
```

Skonfigurujmy teraz dodane pluginy, na końcu pliku *myrest/app/config/config.yml* dopisujemy:

```
fos_rest:
    serializer:
        serialize_null: true
    view:
        view_response_listener: force
        force_redirects:
          html: true
        formats:
            json: true
            xml: true
            rss: false
        templating_formats:
            html: true
        mime_types:
                                  json:   ['application/json',   'application/x-json',
'application/vnd.example-com.foo+json']
            rss: 'application/rss+xml'
            jpg: 'image/jpeg'
            png: 'image/png'
    body_listener: true
    param_fetcher_listener: force
    allowed_methods_listener: true
    access_denied_listener:
        json: true
    format_listener:
        rules:
            - { path: '^/', priorities: ['html','json', 'xml'], fallback_format:
html, prefer_extension: true }
    routing_loader:
        default_format: ~
    exception:
        codes:
            'Symfony\Component\Routing\Exception\ResourceNotFoundException': 404
        messages:
            'Symfony\Component\Routing\Exception\ResourceNotFoundException': true
```

Czas stworzyć pierwszy moduł. W tym celu należy utworzyć nowy pakiet:

```
php app/console generate:bundle --namespace=Rest/DemoBundle --dir=src --no-
interaction
```

Kolejnym krokiem będzie konfiguracja bazy danych. W tym celu edytujemy plik *myrest/app/config/parameters.yml*:

```
# app/config/parameters.yml
parameters:
    database_driver:    pdo_mysql
    database_host:      localhost
    database_name:      sym_rest
    database_user:      root
    database_password:  password

# ...
```

Odpowiednio zmieniamy wybrane parametry w taki sposób aby były zgodne z ustawieniami naszego serwera bazy danych.

Czas zająć się modelem aplikacji. Poniższym poleceniem utworzymy bazę danych, której nazwę przed chwilę określiliśmy, na naszym serwerze baz danych:

```
php app/console doctrine:database:create
```

Czas utworzyć naszą klasę (ang. Entity class) odwzorowującą strukturę naszego modelu w pliku *myrest/src/Rest/DemoBundle/Entity/DailyNote.php*:

```
<?php

namespace Rest\DemoBundle\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity
 * @ORM\Table(name="dailynotes")
 */
class DailyNote
{
    /**
     * @ORM\Column(type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    protected $id;

    /**
     * @ORM\Column(type="string", length=255)
     */
    protected $imgUrl;

    /**
     * @ORM\Column(type="date")
     */
    protected $date;

    /**
     * @ORM\Column(type="text")
     */
    protected $description;

    /**
     * @ORM\Column(type="integer")
     */
    protected $stars;

    /**
     * @ORM\Column(type="boolean")
     */
    protected $evaluated;
}
```

Poniższym poleceniem wygenerujemy metody dostępu (setery i getery):

```
php app/console doctrine:generate:entities Rest/DemoBundle/Entity/DailyNote
```

Kolejnym poleceniem na podstawie wygenerowanego kodu utworzymy odpowiednią strukturę w bazie danych:

```
php app/console doctrine:schema:update --force
```

Stwórzmy teraz formularz dla naszego modelu:

```
php app/console doctrine:generate:form RestDemoBundle:DailyNote --no-interaction
```

Wprowadźmy pewne wymagane modyfikacje do wygenerowanego pliku *myrest/src/Rest/DemoBundle/Form/DailyNoteType.php*:

```
...
public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('imgUrl', 'text')
              ->add('date', 'date', array( 'widget' => 'single_text', 'format' =>
'yyyy-MM-dd',))
            ->add('description', 'textarea')
            ->add('stars', 'number')
            ->add('evaluated', 'checkbox')
        ;
    }
...
public function setDefaultOptions(OptionsResolverInterface $resolver)
    {
        $resolver->setDefaults(array(
            'data_class' => 'Rest\DemoBundle\Entity\DailyNote',
            'csrf_protection'   => false
        ));
    }
...
```

Wprowadzane za pośrednictwem formularza dane muszą przejść przez walidację, której warunki określamy w pliku *myrest/src/Rest/DemoBundle/Resources/config/validation.yml* (utwórz nowy plik):

```
Rest\DemoBundle\Entity\DailyNote:
    properties:
        imgUrl:
            - NotBlank: ~
            - NotNull: ~
            - Length:
               min: 2
               max: 255
               minMessage: "The image url must be at least  characters length"
               maxMessage: "The image url cannot be longer than  characters length"
        date:
            - NotBlank: ~
            - NotNull: ~
            - Date: ~
        description:
            - Type:
                type: string
        stars:
            - NotBlank: ~
            - NotNull: ~
            - Range:
                min: 0
                max: 5
                minMessage: You can set minimum 0 stars
                maxMessage: You can set minimum 5 stars
        evaluated:
            - Type:
                type: boolean
```

Załóżmy, że chcemy udostępniać nasze api pod adresem /api/. Tu będziemy mieć zasób, który pozwoli na czytanie (GET), zapisywanie (POST), zmianę (PUT) i kasowanie (DELETE) danych. Musimy najpierw zająć się routingiem.

Zajrzyjmy do pliku *myrest/app/config/routing.yml* i wprowadźmy następujące zmiany:

```
rest_demo:
    type: rest
```

```
    prefix: /api
    resource: "@RestDemoBundle/Resources/config/routing.yml"
```

Następnie tworzymy plik *myrest/src/Rest/DemoBundle/Resources/config/routing.yml* dla naszego bundle'a i umieszczamy tam definicję:

```
rest_demo_DailyNote:
    type: rest
    prefix: /v1
    resource: "Rest\DemoBundle\Controller\DailyNoteController"
    name_prefix:  api_1_ # naming collision
```

Teraz, gdy mamy już utworzony model zarówno w aplikacji, jak i w bazie danych, możemy przejść do napisania pierwszego kontrolera, który będzie obsługiwał metodę GET. W tym celu utwurzmy nowy kontroler *DailyNoteController* i dodajemy metodę getAllAction() w klasie tego kontrolera w pliku *mysert/src/Rest/DemoBundle/Controller/DailyNoteController.php*:

```
<?php

namespace Rest\DemoBundle\Controller;

use FOS\RestBundle\Controller\FOSRestController;

class DailyNoteController extends FOSRestController {
    public function postNoteAction() { }
}
```

Nie dziedziczymy po zwykłym kontrolerze Symfony. *FOSRestController* jest naszą bazą do tworzenia kontrolerów. Dzięki niemu wystarczy dodać odpowiednie akcje aby routing zadziałał. Sprawdźmy jak wyglądają teraz trasy:

```
php app/console router:debug
```

Powinno nam się wyświetlić m.in.:

...

api_1_post_note POST ANY ANY /api/v1/notes.{_format}

...

Mamy teraz naszą metodę. Ale skąd wzięło get i notes? Typ routingu „rest” znaczy, że symfony skanuje nasz kontroler szukając akcji. Każdą akcję rozkłada na czynniki pierwsze. Nasza „getNotesAction” stała się metodą HTTP „get”. Z kolei „notes” – jest nazwą zasobu. REST bundle sam dokleja parametr „_format”, który mówi jak zwrócić dane.

Dopiszmy kolejne metody aby zobaczyć jak zmienią się nasze trasy (*mysert/src/Rest/DemoBundle/Controller/DailyNoteController.php*):

```
<?php
...
use Symfony\Component\HttpFoundation\Request;
...
class DailyNoteController extends FOSRestController {
    public function getNotesAction(Request $request) { }
    public function getNoteAction($id) { }
    public function postNoteAction(Request $request) { }
    public function putNoteAction(Request $request, $id) { }
    public function deleteNoteAction(Request $request, $id) { }
}
```

Powinno nam się wyświetlić m.in.:

…

api_1_get_notes	GET	ANY	ANY	/api/notes.{_format}
api_1_get_note	GET	ANY	ANY	/api/notes/{id}.{_format}
api_1_post_note	POST	ANY	ANY	/api/notes.{_format}

api_1_put_note	PUT	ANY	ANY	/api/notes/{id}.{_format}
api_1_delete_note	DELETE	ANY	ANY	/api/notes/{id}.{_format}

…

Opcjonalny parametr *_format* pozwala wybrać, w jakim formacie będą dane zwrócone przez nasze API. Same ścieżki i puste funkcje kontrolera na nic nam się zdarzą, pora dodać logikę sterowania i zwrócić konkretne dane (*mysert/src/Rest/DemoBundle/Controller/DailyNoteController.php*):

```
<?php
namespace Rest\DemoBundle\Controller;

use Symfony\Component\HttpFoundation\Request;

use FOS\RestBundle\Controller\FOSRestController;
use FOS\RestBundle\Util\Codes;
use FOS\RestBundle\Controller\Annotations;

use Rest\DemoBundle\Exception\InvalidFormException;
use Rest\DemoBundle\Form\DailyNoteType;
use Rest\DemoBundle\Model\DailyNoteInterface;

class DailyNoteController extends FOSRestController
{
    /**
     * List all notes.
     *
     * @Annotations\View(
     *  templateVar="notes"
     * )
     *
    * @return array
    */
    public function getNotesAction()
    {
            return $this->container->get('rest_demo.dailynote.handler')->all();
    }

    public function getNoteAction($id) { }

    /**
     * Create a DailyNote from the submitted data.
     *
     * @Annotations\View(
     *  statusCode = Codes::HTTP_BAD_REQUEST
     * )
     *
     * @return FormTypeInterface|RouteRedirectView
     */
    public function postNoteAction(Request $request)
    {
        try {
            $newNote = $this->container->get('rest_demo.dailynote.handler')->post(
                    $request->request->all()
            );
            $routeOptions = array(
                'id' => $newNote->getId(),
                '_format' => $request->get('_format')
            );
                 return  $this->routeRedirectView('api_1_get_note',  $routeOptions,
Codes::HTTP_CREATED);
        } catch (InvalidFormException $exception) {
            return array('form' => $exception->getForm());
        }
```

```
    }

    public function putNoteAction(Request $request, $id) { }
    public function deleteNoteAction(Request $request, $id) { }
}
```

Teraz dopiszemy wymagane interfejsy i serwisy (obiekty obsługujące modyfikację danych w modelach). Zgodnie z dobrymi praktykami programowania obiektowego musimy stworzyć w pliku *myrest/src/Rest/DemoBundle/Model/DailyNoteInterface.php* interfejs dla naszego modelu *DailyNote*:

```
<?php

namespace Rest\DemoBundle\Model;

interface DailyNoteInterface {

    /**
     * Set imgUrl
     *
     * @param string $imgUrl
     * @return DailyNoteInterface
     */
    public function setImgUrl($imgUrl);

    /**
     * Get imgUrl
     *
     * @return string
     */
    public function getImgUrl();

    /**
     * Set date
     *
     * @param \DateTime $date
     * @return DailyNoteInterface
     */
    public function setDate($date);

    /**
     * Get date
     *
     * @return \DateTime
     */
    public function getDate();

    /**
     * Set description
     *
     * @param string $description
     * @return DailyNoteInterface
     */
    public function setDescription($description);

    /**
     * Get description
     *
     * @return string
     */
    public function getDescription();

    /**
     * Set stars
     *
```

```
     * @param integer $stars
     * @return DailyNoteInterface
     */
    public function setStars($stars);

    /**
     * Get stars
     *
     * @return integer
     */
    public function getStars();

    /**
     * Set evaluated
     *
     * @param boolean $evaluated
     * @return DailyNoteInterface
     */
    public function setEvaluated($evaluated);

    /**
     * Get evaluated
     *
     * @return boolean
     */
    public function getEvaluated();
}
```

Zmodyfikujmy teraz plik *myrest/src/Rest/DemoBundle/Entity/DailyNote.php*:

```
<?php

namespace Rest\DemoBundle\Entity;

use Doctrine\ORM\Mapping as ORM;
use Rest\DemoBundle\Model\DailyNoteInterface;

/**
 * @ORM\Entity
 * @ORM\Table(name="dailynotes")
 */
class DailyNote implements DailyNoteInterface
{
```

Utwórzmy teraz klasę obsługującą żądania modyfikacji modelu *DailyNote* – klasę *DailyNoteHandler* w pliku *myrest/src/Rest/DemoBundle/Handler/DailyNoteHandler.php*:

```
<?php
namespace Rest\DemoBundle\Handler;

use Doctrine\Common\Persistence\ObjectManager;
use Symfony\Component\Form\FormFactoryInterface;

use Rest\DemoBundle\Model\DailyNoteInterface;
use Rest\DemoBundle\Form\DailyNoteType;
use Rest\DemoBundle\Exception\InvalidFormException;

class DailyNoteHandler implements DailyNoteHandlerInterface
{
    private $om;
    private $entityClass;
    private $repository;
    private $formFactory;

    public function __construct(ObjectManager $om, $entityClass,
                                            FormFactoryInterface $formFactory)
```

```
    {
        $this->om = $om;
        $this->entityClass = $entityClass;
        $this->repository = $this->om->getRepository($this->entityClass);
        $this->formFactory = $formFactory;
    }

    /**
     * Create a new DailyNote.
     *
     * @param array $parameters
     *
     * @return DailyNoteInterface
     */
    public function post(array $parameters)
    {
        $note = $this->createNote();

        return $this->processForm($note, $parameters, 'POST');
    }

    /**
     * Processes the form.
     *
     * @param DailyNoteInterface $note
     * @param array         $parameters
     * @param String        $method
     *
     * @return DailyNoteInterface
     *
     * @throws \Rest\DemoBundle\Exception\InvalidFormException
     */
    private function processForm(DailyNoteInterface $note, array $parameters,
                                                            $method = "PUT")
    {
        $form = $this->formFactory->create(new DailyNoteType(), $note,
                                                array('method' => $method));
        $form->submit($parameters, 'PATCH' !== $method);

        if ($form->isValid()) {

            $note = $form->getData();
            $this->om->persist($note);
            $this->om->flush($note);

            return $note;
        }

        throw new InvalidFormException('Invalid submitted data', $form);
    }

     /**
      * Get a list of Pages.
      *
      * @return array
      */
     public function all()
     {
            return $this->repository->findAll();
     }

     private function createNote()
     {
         return new $this->entityClass();
```

```
    }
}
```

Powyższa klasa *DailyNoteHandler* implementuje interfejs, który musimy stworzyć w pliku *myrest/src/Rest/DemoBundle/Handler/DailyNoteHandlerInterface.php*:

```
<?php
namespace Rest\DemoBundle\Handler;

use Rest\DemoBundle\Model\DailyNoteInterface;

Interface DailyNoteHandlerInterface
{
    /**
     * Get a DailyNote given the identifier
     *
     * @api
     *
     * @param mixed $id
     *
     * @return DailyNoteInterface
     */
    public function get($id);

    /**
     * Post DailyNote, creates a new DailyNote.
     *
     * @param array $parameters
     *
     * @return DailyNoteInterface
     */
    public function post(array $parameters);
}
```

Klasę *DailyNoteHandler* musimy udostępnić jako serwis (wstrzykiwanie zależności), w tym celu edytujemy plik *myrest/src/Rest/DemoBundle/Resources/config/services.xml:*

```
<?xml version="1.0" ?>
<container xmlns="http://symfony.com/schema/dic/services"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://symfony.com/schema/dic/services
                        http://symfony.com/schema/dic/services/services-1.0.xsd">
    <parameters>
        <parameter key="rest_demo.dailynote.handler.class">
            Rest\DemoBundle\Handler\DailyNoteHandler
        </parameter>
        <parameter key="rest_demo.dailynote.class">
            Rest\DemoBundle\Entity\DailyNote</parameter>
       </parameters>

      <services>
            <service id="rest_demo.dailynote.handler"

class="%rest_demo.dailynote.handler.class%">
                  <argument type="service" id="doctrine.orm.entity_manager" />
                  <argument>%rest_demo.dailynote.class%</argument>
                  <argument type="service" id="form.factory"></argument>
            </service>
      </services>
</container>
```

Pozostała już tylko do napisania klasa definiująca rzucany przez funkcję processForm() wyjątek – InvalidFormException (*myrest/src/Rest/DemoBundle/Exception/InvalidFormException.php):*

```
<?php

namespace Rest\DemoBundle\Exception;
```

```
class InvalidFormException extends \RuntimeException
{
    protected $form;

    public function __construct($message, $form = null)
    {
        parent::__construct($message);
        $this->form = $form;
    }

    /**
     * @return array|null
     */
    public function getForm()
    {
        return $this->form;
    }
}
```

Ponownie przetestujmy aplikację:

```
php app/console server:run
```

a następnie w nowym oknie/zakładce konsoli wpiszmy:

```
curl -X POST -d '{"imgUrl": "img/smile1.png", "date": "2014-04-12", "description":
"test", "stars": 3, "evaluated": true}' http://localhost:8000/api/v1/notes.json
--header "Content-Type:application/json" -v
```

w wyniku czego otrzymamy:

* Hostname was NOT found in DNS cache
*   Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 8000 (#0)
> POST /api/v1/notes.json HTTP/1.1
> User-Agent: curl/7.35.0
> Host: localhost:8000
> Accept: */*
> Content-Type:application/json
> Content-Length: 104
>
* upload completely sent off: 104 out of 104 bytes
< HTTP/1.1 201 Created
< Host: localhost:8000
< Connection: close
< X-Powered-By: PHP/5.5.9-1ubuntu4.7
< Cache-Control: no-cache
< Date: Mon, 20 Apr 2015 14:58:59 GMT
< Location: http://localhost:8000/api/v1/notes/7.json
< Allow: POST, GET
< Content-Type: application/json
< X-Debug-Token: fe4156
< X-Debug-Token-Link: /_profiler/fe4156

<
* Closing connection 0

oraz

```
curl -i -H "Accept: application/json"  localhost:8000/api/v1/notes -v
```

po czym otrzymamy:

* Hostname was NOT found in DNS cache
*   Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 8000 (#0)
> GET /api/v1/notes HTTP/1.1
> User-Agent: curl/7.35.0
> Host: localhost:8000
> Accept: application/json
>
< HTTP/1.1 200 OK
< Host: localhost:8000
< Connection: close
< X-Powered-By: PHP/5.5.9-1ubuntu4.7
< Cache-Control: no-cache
< Date: Mon, 20 Apr 2015 15:00:24 GMT
< Content-Type: application/json
< Allow: POST, GET
< X-Debug-Token: cedc62
< X-Debug-Token-Link: /_profiler/cedc62
<
* Closing connection 0
[{"id":1,"img_url":"img\/smile1.png","date":"2014-04-12T00:00:00+0200","description":"test","stars":3,"evaluated":true}]

Zadanie 4 Integracja z aplikacją kliencką

W tym zadaniu wykorzystamy aplikację z poprzednich laboratoriów. Dokonamy jednak refactoringu, którzy pozwoli nam lepiej wykorzystać możliwości AngularJS. Zacznijmy od stworzenia podstawowej struktury folderów dla aplikacji Kalendarz:

```
\kalendarz
| - \css
| - \js
| - \partials
| - \img
```

W katalogu *kalendarz/css* umieszczamy plik *style.css* (z poprzedniego laboratorium). W katalogu *kalendarz/img* powinny znaleźć się wszystkie obrazki z katalogu *img* (z poprzedniego laboratorium).

Wewnątrz katalogu *kalendarz/js* tworzymy pliki *app.js*, *services.js*, *controllers.js* i *directives.js*.

W pliku *kalendarz/js/app.js* umieszczamy:

```
'use strict';

/* Modules */
```

```
var app = angular.module('calendar', [
      'noteControllers',
      'noteDirectives',
      'noteServices'
      ]);
```

W pliku *kalendarz/js/services.js* umieszczamy:

```
'use strict';

/* Services */

var noteServices = angular.module('noteServices', ['ngResource']);

noteServices.factory('Note', ['$resource', function($resource){
    return $resource('http://localhost:8000/api/v1/notes/:noteId.json', {}, {
      query: {method:'GET', isArray:true}
      });
}]);
```

W pliku *kalendarz/js/controllers.js* umieszczamy:

```
'use strict';

/* Controllers */

var noteControllers = angular.module('noteControllers', []);

noteControllers.controller('CalendarController', ['$scope', 'Note',
 function($scope, 'Note') {
      this.seriesNo = -1;
      this.series = [Note.query()];

      this.setSeriesNo = function(numb) {
            this.seriesNo = numb;
      };

      this.getSeries = function() {
            return this.series[this.seriesNo];
      };

      $scope.range = function(range) {
            var arr = [];
            for (var i = 0; i < range; i++) {
                  arr += i+1;
            }
            return arr;
      };

      this.addSeries = function(object) {
            var idx = this.series.indexOf(object) + 1;
            this.series.splice(idx, 0, []);
      };
}]);

noteControllers.controller('DropDownListController', ['$scope', function($scope) {
      this.clicked = false;
      $scope.ddlSelected = 'img/smile0.png';
      this.itemsList = [];

      for (var i = 1; i <= 4; i++)
            this.itemsList.push('img/smile'+ i +'.png');

      this.toggle = function() {
            this.clicked = !this.clicked;
```

```
	};

	this.select = function(item) {
		this.clicked = false;
		$scope.ddlSelected = item;
	};
}]);

noteControllers.controller('NoteFormController', ['$scope', 'Note',
 function($scope, Note) {
	$scope.note = new Note();
	$scope.note.stars = 0;

	var getDate = function() {
		var d = new Date();
		var day = d.getDate();
		var mth = d.getMonth()+1;
		var year = d.getFullYear().toString();
		day = (day > 9) ? day.toString() : '0'+day;
		mth = (mth > 9) ? mth.toString() : '0'+mth;
		return year + '-' + mth + '-' + day;
	};

	this.init = function() {
		$scope.note = new Note();
		$scope.note.stars = 0;
	};

	this.submit = function(series) {
		$scope.note.date = getDate();
		$scope.note.imgUrl = $scope.ddlSelected;
		Note.save($scope.note, function() {
			series.push($scope.note);
			$scope.note = new Note();
			$scope.note.stars = 0;
		});
	};
}]);
```

W pliku *kalendarz/js/directives.js* umieszczamy:

```
'use strict';

/* Directives */

var noteDirectives = angular.module('noteDirectives', []);

noteDirectives.directive('dropDownList', function() {
	return {
		restrict: 'A',
		templateUrl: 'partials/drop-down-list.html',
		controller: 'DropDownListController',
		controllerAs: 'ddlCtrl'
	};
});

noteDirectives.directive('noteForm', function() {
	return {
		restrict: 'E',
		templateUrl: 'partials/note-form.html',
		controller: 'NoteFormController',
		controllerAs: 'formCtrl'
	};
});
```

Utwórzmy teraz plik *kalendarz/index.html*:

```
<!DOCTYPE html>
<html lang="pl" ng-app="calendar">
      <head>
          <title>Kalendarz</title>
          <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
          <link rel="stylesheet" type="text/css" href="css/style.css">
          <script
      src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.15/angular.min.js">
          </script>
          <script
    src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.15/angular-route.js">
          </script>
          <script
  src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.15/angular-resource.js">
          </script>

          <script src="js/app.js"></script>
          <script src="js/services.js"></script>
          <script src="js/controllers.js"></script>
          <script src="js/directives.js"></script>
      </head>
      <body ng-controller='CalendarController as calCtrl'>
          <div class="table">
            <div class="row" ng-repeat='days in calCtrl.series track by $index'>

                  <div class="cell" ng-repeat='day in days'>
                        <h4>{{ $index + 1 }}</h4>
                        <img ng-src="{{ day.img_url || '//:0' }}">
                        <h5>{{ day.date }}</h5>
                        <span>{{ day.description }}</span>
                        <p ng-show='day.evaluated'>
                              <img ng-repeat="x in range(day.stars)"
                                                      ng-src='img/star.png' />
                        </p>
                  </div>
                  <div class="cell">
                        <ul>
                              <li>
                                    <a href="#openModal"
                                                ng-click='setSeriesNo($index)'>
                                          Dodaj wpis
                                    </a>
                              </li>
                              <li>
                                    <a ng-click='addSeries(days)'>
                                          Dodaj serię
                                    </a>
                              </li>
                        </ul>
                  </div>
          </div>
          <div id="openModal" class="modalDialog">
                  <div>
                        <a href="#close" title="Close" class="close">X</a>
                        <note-form></note-form>
                  </div>
          </div>
      </body>
</html>
```

Teraz utwórzmy plik *kalendarz/partials/calendar.html*:

```
<div class="table">
```

```
      <div class="row" ng-repeat='days in series track by $index'>
            <div class="cell" ng-repeat='day in days'>
                  <h4>{{ $index + 1 }}</h4>
                  <img ng-src="{{ day.img_url || '//:0' }}">
                  <h5>{{ day.date }}</h5>
                  <span>{{ day.description }}</span>
                  <p ng-show='day.evaluated'><img ng-repeat="x in range(day.stars)"
ng-src='img/star.png' /></p>
            </div>
            <div class="cell">
                  <modal-box></modal-box>
            </div>
    </div>
</div>
```

Dalej utwórzmy plik *kalendarz/partials/drop-down-list.html*:

```
<span class='sp'>Wybierz obrazek:</span><br />
<div class="selection-wrapper">
      <div class="selected-item-box" ng-click='ddlCtrl.toggle()'>
            <span class="dropdown-icon"></span>
            <ul class="items-list">
                  <li>
                        <img ng-src="{{ ddlSelected }}"/>
                  </li>
            </ul>
      </div>
      <div class="list" ng-show='ddlCtrl.clicked'>
            <ul class="items-list">
                  <li     ng-click='ddlCtrl.select(item)'     ng-repeat='item     in
ddlCtrl.itemsList'>
                        <img ng-src="{{ item }}" />
                  </li>
            </ul>
      </div>
</div>
```

Utwórzmy plik *kalendarz/partials/note-form.html*:

```
<h2>Dodaj nowy wpis</h2>
<form action='#close' ng-submit='formCtrl.submit(getSeries())'
                                                      ng-init='formCtrl.init()'>
      <div class='form-field' drop-down-list ></div>
      <div class='form-field'>
            <span class='sp'>Oceń:</span>
            <input type='checkbox' ng-model='note.evaluated'>
      </div>
      <div class='form-field' ng-show='note.evaluated'>
            <span class='sp'>Ocena:</span>
            <input ng-repeat="x in range(5)" type='radio' ng-model="note.stars"
                                                            value="{{x}}">
      </div>
      <div class='form-field'>
            <span class='sp'>Podaj opis (opcjonalnie):</span><br />
            <textarea ng-model='note.description'></textarea>
      </div>
      <div class='form-field'>
            <input type='submit' value='Dodaj obserwację'>
      </div>
</form>
```

Przetestujmy aplikację.

Zablokowano żądanie do zasobu innego pochodzenia: zasady „Same Origin Policy” nie pozwalają wczytywać zdalnych zasobów z „http://localhost:8000/api/v1/notes.json”.

Można to zmienić przenosząc zasób do tej samej domeny lub korzystając z CORS.

Zadanie 5 Cross Origin Resource Sharing

Testy aplikacji wskazały na problem braku dostępu do zasobów serwera znajdujących się pod inną domeną i/lub portem. W celu poradzenia sobie z tym problemem wymagana jest instalacja i konfiguracja dodatkowego pakietu dla aplikacji RESTowej. W tym celu w konsoli należy przejść do katalogu projektu aplikacji (*myrest*) i zainstalować dodatkowego bundle'a:

```
php ../composer.phar require nelmio/cors-bundle
```

Dalej dodajemy zainstalowany pakiet do jądra aplikacji w pliku *myrest/app/AppKernel.php*:

```
public function registerBundles()
{
    $bundles = array(
        ...
        new Nelmio\CorsBundle\NelmioCorsBundle(),
        ...
    );
    ...
}
```

Dodajemy na końcu pliku konfiguracyjnego (*myrest/app/config/config.yml*) wpis:

```
nelmio_cors:
    defaults:
        allow_credentials: false
        allow_origin: []
        allow_headers: []
        allow_methods: []
        expose_headers: []
        max_age: 0
        hosts: []
        origin_regex: false
    paths:
        '^/api/':
            allow_origin: ['*']
            allow_headers: ['X-Custom-Auth', 'content-type']
            allow_methods: ['POST', 'PUT', 'GET', 'DELETE', 'OPTIONS']
            max_age: 3600
```

Ostatecznie pozostaje już tylko obsłużenie wysyłanej przez przeglądarki inicjującej metody OPTIONS, która sprawdza ustawienia serwera. W tym celu do kontrolera *DailyNoteController* (*myrest/src/Rest/DemoBundle/Controller/DailyNoteController.php*) dodajemy funkcje *optionsNotesAction()*:

```
...
public function optionsNotesAction()
{
      return null;
}
...
```

W ten sposób kontroler dla metody OPTIONS będzie zwracał kod odpowiedzi 200 OK.

Aplikacja Kalendarz również wymaga konfiguracji. W tym celu w pliku kalendarz/js/app.js dopisujemy:

```
app.config(['$httpProvider', function($httpProvider) {
  $httpProvider.defaults.useXDomain = true;
  $httpProvider.defaults.headers.post['Accept'] =
                                              'application/json, text/javascript';
  $httpProvider.defaults.headers.post['Content-Type'] =
                                               'application/json; charset=utf-8';
```

```
  delete $httpProvider.defaults.headers.common['X-Requested-With'];
}]);
```

Ponownie przetestujmy aplikację.

Zadanie 6 Implementacja dodatkowych funkcjonalności

W ramach samodzielnej pracy należy zaimplementować funkcjonalność:

a) edycji wpisów
b) usuwania wpisów
c) obsługa WebStorage dla wczytywania, edycji i usuwania danych (po stronie aplikacji klienta),
d) autoryzacji żądań,
e) obsługi serii,
f) obsługi walidacji formularza,
g) itp.