CODING WITH DRUPAL 8

Drupalcamp Vienna 2015/11

by Wolfgang Ziegler / @the_real_fago

ABOUT ME

- Wolfgang Ziegler
- drupal.org/u/fago
- @the_real_fago
- CEO of drunomics

WE ARE HIRING!

- PHP / Drupal Backend EntwicklerIn
- PHP / Drupal WebentwicklerIn

drunomics

MY CONTRIBUTIONS

- Diverse modules since ca. 2006
  - Rules
  - Entity API
  - Field-collection
  - Profile2
- Drupal core subsystem maintainer
  - Form system
  - Entity API
  - Typed Data API
- Leading force behind the Drupal 8 Entity API improvements

#D8RULES

see d8rules.org

Drupal™

A NEW DRUPAL!

- Same concepts, new code
- Useful out-of-the box
- Ready: 8.0.0

SOME FEATURES (EXCERPT..)

- Improved authoring experience
- Fully translatable
- Configuration management
- Views in core
- Fixed theming (twig)

DRUPAL CODING HISTORY

DRUPAL <= 7

- Prozedural, PHP 4 style
- Funktionen, Arrays
- Hooks -> Modularität

HOOKS

modules/some_module/some_module.module:

```
$info = module_invoke_all('node_type_info', $some_arg);
```

modules/another_module/another_module.module:

```
function ANOTHER_MODULE_node_type_info($some_arg) {
  return array('some_type' => array('label' => 'example')));
}
```

IMPLEMENTATION

of forms, blocks or menu items, ...
via

- Arrays
- Callbacks
- Nested Arrays
- Deeeeeply nested arrays

CODING WITH DRUPAL 7

- Easy to get start
- but under the hood:
  hard to grasp

PHP RENAISSANCE

- PHP Framework Interopaerability Group (FIG)
- PSR-0, PSR-4 Autoloading Standards
- PHP packages
- Active development of PHP
- Performance improvements
  (PHP-7, HHVM)

NAMESPACES

- since PHP 5.3
- Foundation for packages
  - Symfony\Component\HttpFoundation
  - Doctrine\Common\Annotations
  - Drupal\Core\Entity

USING NAMESPACES

```php
<?php

namespace MyProject\Component;

use Symfony\Component\HttpFoundation\Request;

class SomeClass {

  public function someMethod(Request $request) {

    if ($request->getPort() != 80) {
      throw new \LogicException("Application may work on port 80 only.");
    }

  }

}
```

COMPOSER

- Dependency manager for PHP
- Install, Use and publish packages
- Package/project metadata: composer.json
- Project-metadata: composer.json
- Project packages: composer.lock

composer.json

```
{
  "name": "commerceguys/pricing",
  "description": "Pricing",
  "license": "MIT",
  "require": {
    "commerceguys/intl": "~0.5"
  },
  "autoload": {
   "psr-4": {
     "CommerceGuys\\Pricing\\": "src"
   }
  },
  "authors": [
   {
      "name": "Bojan Zivanovic"
    }
  ]
```

FINDING PACKAGES?

- packagist.org
- packagist.drupal-composer.org

DRUPAL MODULES ARE PACKAGES DEPENDING ON CORE!

USING IT

```
composer install
```

```
composer create-project drupal-composer/drupal-project:8.x-dev
```

```
composer require drupal/devel:8.*
```

CODING WITH DRUPAL 8

- getting off the island
- object oriented
- OOP basics are necessary

PACKAGES IN DRUPAL 8

- Symfony components (...)
- Twig
- PHPUnit
- Guzzle
- ZendFeed
- Doctrine (Annotations)

LEARNING CODING WITH DRUPAL 8 IS...

LEARNING PHP CODING!

PHP BEST PRACTICES

http://www.phptherightway.com

DRUPAL 8:
THE ESSENTIALS!

OVERVIEW

- Modules
- Controller, Routing & Menus
- Plugin system
- Entities & Fields
- Configuration system

MODULES

DIRECTORY STRUCTURE

```
core/modules
    ├── action
    ├── aggregator
    ...

modules/some_module/
    ├── config
    ├── css
    ├── js
    ├── src
    │   └── Entity
    │       └── SomeEntity.php
    ├── templates
    ├── tests
    ├── some_module.info.yml
    └── some_module.module
```

some_module.info.yml

```yaml
name: Some module
type: module
description: 'A description, usually in English.'
package: Some
core: 8.x
dependencies:
  - text
  - entity_reference
```

DIRECTORY STRUCTURE

```
core/modules
├── action
├── aggregator
...

modules/some_module/
├── config
├── css
├── js
├── src
│   └── Entity
│       └── SomeEntity.php
├── templates
├── tests
├── some_module.info.yml
└── some_module.module
```

some_module/src/Entity/SomeEntity.php

```php
<?php

/**
 * @file
 * Contains \Drupal\some_module\Entity\SomeEntity.
 */

namespace Drupal\some_module\Entity;

use Drupal\Core\Entity\ContentEntityBase;

/**
 * Some entity.
 *
 * ...
 */
class SomeEntity extends ContentEntityBase {
  //...
```

MODULE INTERACTION

- Services, Plugins
- Events (Symfony)
- Hooks

CONTROLLER, ROUTING & MENUS

CONTROLLER

- Part of the MV**C** module
- Captures the behavior of the app
- Controls the ouput / HTTP response

EXAMPLE: USERCONTROLLER

```php
<?php

class UserController extends ControllerBase {

  /**
   * Logs the current user out.
   *
   * @return \Symfony\Component\HttpFoundation\RedirectResponse
   *   A redirection to home page.
   */
  public function logout() {
    user_logout();
    return $this->redirect('<front>');
  }
}
```

Which controller is in charge of

`/path/123?example=1`

?

ROUTING

- based on Symfony component(s)
- HTTP foundation replace `$_GET`, `$_POST`, etc.
- Replacement: `$request` & `$response` objects

core/modules/user/user.routing.yml

```
user.page:
  path: '/user'
  defaults:
    _controller: '\Drupal\user\Controller\UserController::userPage'
    _title: 'My account'
  requirements:
    _user_is_logged_in: 'TRUE'
```

core/modules/user/user.routing.yml

```yaml
entity.user.canonical:
  path: '/user/{user}'
  defaults:
    _entity_view: 'user.full'
    _title_callback: 'Drupal\user\Controller\UserController::userTitle'
  requirements:
    _entity_access: 'user.view'
```

MENU SYSTEM

core/modules/user/user.links.menu.yml

```yaml
user.page:
  title: 'My account'
  weight: -10
  route_name: user.page
  menu_name: account
```

PLUGIN SYSTEM

PLUGINS?

- small functional units
- a plugin class
- defined interface
- exchangeable
- different plugin types, e.g.:
  - Field types (text, integer, image..)
  - Field widgets & formatters
  - Blocks

PLUGINS & MODULES

- Modules can provide plugins
- Modules can define plugin types
- Replace lots of hooks

IMPLEMENTING A PLUGIN

- Adding a php class (in the proper directory)
- Implementing an interface
- Adding an annotation

EXAMPLE: PROVIDE A BLOCK

- `BlockPluginInterface`
- extend `BlockBase`

../search/src/Plugin/Block/SearchBlock.php

```php
/**
 * @Block(
 *   id = "search_form_block",
 *   admin_label = @Translation("Search form"),
 *   category = @Translation("Forms")
 * )
 */
class SearchBlock extends BlockBase {

  protected function blockAccess(AccountInterface $account) {
    return AccessResult::allowedIfHasPermission($account, 'search content');
  }
  public function build() {
    return $this->formBuilder->getForm('Drupal\search\Form\SearchBlockForm');
  }
}
```

ADVANTAGES OF PLUGINS

- self-contained
- clearly documented (interfaces)
- class autoloading

ENTITIES & FIELDS

Drupal's data model

ENTITY SYSTEM

- replaces direct database access
- CRUD und Query APIs
- Storage in-dependent

ENTITY CRUD

```php
$node = Node::create(array(
  'type' => 'page',
  'title' => 'Example',
));
$node->save();


$id = $node->id();
$node = Node::load($id);


$node->delete();
```

ENTITY QUERY

```
$term_ids = \Drupal::entityQuery('taxonomy_term')
  ->condition('vid', $vocabulary_id)
  ->execute();
```

ENTITIES

Content

+

Configuration

CONFIGURATION ENTITIES

- Stored as configuration
- Simple objects with properties
- Node types, Views, Image styles, ...

CONTENT ENTITIES

- Fieldable
- Revisionable
- Translatable
- Content nodes, Users, Comments, ...

CONTENT ENTITIES

- Consist only of fields
- Field values are objects
- Configuration for view/form display
- Fields controlled by code or config

../comment/src/Entity/Comment.php

```
/**
 * @ContentEntityType(
 *   id = "comment",
 *   label = @Translation("Comment"),
 *   handlers = {
 *     "storage" = "Drupal\comment\CommentStorage",
 *     "access" = "Drupal\comment\CommentAccessControlHandler",
 *     "view_builder" = "Drupal\comment\CommentViewBuilder",
 *     "form" = {
 *       "default" = "Drupal\comment\CommentForm",
 *     },
 *   },
 *   base_table = "comment",
 *   data_table = "comment_field_data",
 *   .....
*/
class Comment extends ContentEntityBase implements CommentInterface {
```

../comment/src/Entity/Comment.php

```php
class Comment extends ContentEntityBase implements CommentInterface {
  public static function baseFieldDefinitions($entity_type) {

    $fields['mail'] = BaseFieldDefinition::create('email')
        ->setLabel(t('Email'))
        ->setDescription(t("The comment author's email address."))
        ->setTranslatable(TRUE);
    $fields['uid'] = BaseFieldDefinition::create('entity_reference')
        ->setLabel(t('User ID'))
        ->setDescription(t('The user ID of the comment author.'))
        ->setTranslatable(TRUE)
        ->setSetting('target_type', 'user')
        ->setDefaultValue(0);

    return $fields;
  }
}
```

ENTITIES IN DRUPAL 8

16:30

by Christophe Galli and Sascha Grossenbacher in room webshapers

CONFIGURATION SYSTEM

CONFIGURATION

- Content types, fields
- Entity displays, widgets, formatters
- Views, blocks
- System settings (Cache, ..)
- ...

FUNCTIONALITY

- Simple import / export (YAML)
- Site config import/export
- fully translatable (Config schema)
- Default-config for modules and distributions

book.settings.yml

```yaml
allowed_types:
  - book
block:
  navigation:
    mode: 'all pages'
child_type: book
```

core/modules/book/config/*

```
├── install
│   ├── book.settings.yml
│   ├── core.base_field_override.node.book.promote.yml
│   ├── core.entity_view_mode.node.print.yml
│   └── node.type.book.yml
└── schema
    └── book.schema.yml
```

book/config/schema/book.schema.yml

```yaml
book.settings:
  type: mapping
  label: 'Book settings'
  mapping:
    allowed_types:
      type: sequence
      label: 'Content types allowed in book outlines'
      sequence:
        - type: string
          label: 'Content type'
    block:
      type: mapping
      label: 'Block'
      mapping:
        navigation:
          type: mapping
          label: 'Navigation'
```

CONFIG API

```
$config = \Drupal::config('book.settings');
$allowed_types = $config->get('allowed_types');

$config = \Drupal::configFactory()->getEditable('book.settings')
$config->set('allowed_types', array('page', 'book'));
$config->save();
```

TESTS

- Unit tests (PHPUnit)
- Integration tests (PHPUnit)
- Functional tests (Simpletest, behat?)

DOCUMENTATION

- https://www.drupal.org/developing/api/8
- https://api.drupal.org/api/drupal/8
- Diverse blog posts (Google)
- Examples in Drupal 8 core (tests!)

DRUPALCONSOLE.COM

QUESTIONS?

THANK YOU

by Wolfgang Ziegler // fago