Designing Beautiful APIs

Shopware Community Day 2013

Tobias Schlitt (@tobySen)

June 11, 2013

About me

- Tobias Schlitt (**Toby**)
- Degree in computer sience
- Proessional PHP since 2000
- Open source enthusiast
- Passion for
  - Software Design
  - Automated Testing

Co-founder of

Helping teams to create high quality web application development.

`http://qafoo.com`

- Expert consulting
- Individual training

Outline

On the beauty of APIs

Getting into code

Summary

Outline

On the beauty of APIs

Getting into code

Summary

A beautiful API . . .

"A beautiful API enables and supports you to realize even unforseen features with elegant code."

–Tobias Schlitt

A beautiful API . . .

- enables you to solve a task with few effort
- keeps you flexible for new features
- is open for extension and adjustment
- bewares of errors
- supports readable and testable code

So, where . . .

Where do you find APIs?

Where do you find APIs?

- Web services
  - REST
  - SOAP
  - Not covered here
- Code
  - Libraries / frameworks
  - Layer / components / modules boundaries
  - Interfaces (and abstract classes!)
  - . . . every single class?!

Levels of Beauty

- Code beauty (of the API)
  - OOP / OOD
  - Consistent, good looking code
  - Tests
- Syntactic / semantical beauty
  - Code structure
  - Naming & meaning
  - Discourage errors
- Documentation
  - API docs
  - Examples & Tutorials
  - User generated docs (comments)
  - Not further examined here

Outline

On the beauty of APIs

Getting into code

Summary

Scenario

- Component for
  - Current project
  - Later projects
  - May become OSS
- Search component
  - Index arbitrary objects
  - Search and retrieve objects
  - Apply filters to results

Step 1

Step 1: Usage

```
1  <?php
2
3  use qafoo\Search\Search;
4
5  // Index
6
7  $product = new Product( 'Glow Stone Driveway' );
8  Service::get()->add( $product );
9
10 // Search
11
12 $products = Search::get()->fetch(
13     'type = "Product" AND name LIKE "%stone%"'
14 );
```

Step 1: API

```
1  namespace qafoo\Search;
2  class Service
3  {
4      public static function get()
5      {}
6      public function fetch( $search, array $config = array() )
7      {}
8      public function suggest( $prefix, array $fields )
9      {}
10     public function add( $object )
11     {}
12 }
```

Step 1: Issues

- Indescriptive
- Ambiguous

Keep in Mind

- Find descriptive names
- Know your audience
- Start with usage examples!

Step 2

Step 2: Usage

```php
<?php

use qafoo\Search\Search;

// Index

$product = new Product( 'Glow Stone Driveway' );
Search::getInstance()->index( $product );

// Search

$products = Search::getInstance()->find(
    'type = "Product" AND name LIKE "%stone%"'
);
```

Step 2: API

```php
namespace qafoo\Search;
class Search
{
    public static function getInstance()
    {}
    public function find( $searchQuery, array $config = array() )
    {}
    public function suggest( $prefix, array $fields )
    {}
    public function index( $object )
    {}
}
```

Step 2: Issues

- Not configurable
- Not easily mockable
- **STATIC DEPENDENCY**

Keep in Mind

- Don’t force to violate Law of Demeter (LoD)
- Leave instancitation / configuration to the user
- **DON’T FORCE STATIC DEPENDENCIES**

Step 3

Step 3: Usage

```php
<?php

use qafoo\Search\Search;

// Index

$search = new Search();
$product = new Product( 'Glow Stone Driveway' );
$search->index( $product );

// Search

$search = new Search();
$products = $search->find(
    'type = "Product" AND name LIKE "%stone%"'
);
```

Step 3: API

```php
namespace qafoo\Search;
class Search
{
    public function __construct()
    {}
    public function find( $searchQuery, array $config = array() )
    {}
    public function suggest( $prefix, array $fields )
    {}
    public function index( $object )
    {}
}
```

Step 3: Issues

- Search & index different concerns?
- Not easily replaceable
- Missing abstraction

Keep in Mind

- Make implementations replaceable
- Stay flexible for changing requirements
- See concerns on different levels

Step 4

Step 4: Usage

```php
use qafoo\Search\Search;

// Object construction, should take place in DIC!
$search = new Search(
    new Backends\SolrSearchBackend( /* ... */ ),
    new Backends\ZeroMqIndexBackend( /* ... */ )
);

// Index
$product = new Product( 'Glow Stone Driveway' );
$search->index( $product );

// Search
$products = $search->find(
    'type = "Product" AND name LIKE "%stone%"'
);
```

Step 4: API

```
1  namespace qafoo\Search;
2  class Search
3  {
4      public function __construct( SearchBackend $searchBackend,
           IndexBackend $indexBackend )
5      {}
6      public function find( $searchQuery, array $config = array() )
7      {}
8      public function suggest( $prefix, array $fields )
9      {}
10     public function index( $object )
11     {}
12 }
```

Step 4: Issues

- Back ends use different syntax
- Writing strings in code sucks
- Query parsing is dedicated concern!

Keep in Mind

- Avoid code is strings
- Separate concerns

Step 5

Step 5: Usage

```php
use qafoo\Search\Search;
use qafoo\Search\Criterion;

// Object construction
$search = new Search( /* ... */ );

// Search
$products = $search->find(
  new Criterion(
    new Criterion( 'type', Criterion::EQUALS, 'Product' ),
    Criterion::L_AND,
    new Criterion( 'name', Criterion::LIKE, '%stone%' )
  )
);
```

Step 5: API

```
1  class Search
2  {
3      public function __construct( SearchBackend $searchBackend,
           IndexBackend $indexBackend )
4      {}
5      public function find( Criterion $criterion, array $config =
           array() )
6      {}
7      public function suggest( $prefix, array $fields )
8      {}
9      public function index( $object )
10     {}
11 }
```

Step 5: API

```
1  namespace qafoo\Search;
2  class Criterion
3  {
4      const EQUALS = '=';
5      const LIKE = 'LIKE';
6      const L_AND = '&&';
7      // ...
8
9      public function __construct( $first, $operator, $second )
10     {}
11 }
```

Step 5: API

```
1  namespace qafoo\Search;
2  class UserQueryParser
3  {
4      public function parseQuery( $queryString )
5      {}
6  }
```

Step 5: Issues

- Criterions hardly extensible
- Parent ctor should always be valid

Keep in Mind

- Never use constants for processing instructions
- Put trees where trees belong . . .

Step 6

Step 6: Usage

```
1  use qafoo\Search\Search;
2  use qafoo\Search\Criterion\LogicalAnd;
3  use qafoo\Search\Criterion\Equals;
4  use qafoo\Search\Criterion\Like;
5
6  // Object construction
7  $search = new Search( /* ... */ );
8
9  // Search
10 $products = $search->find(
11   new LogicalAnd(
12     new Equals( 'type', 'Product' ),
13     new Like( 'name', '%stone%' )
14   )
15 );
```

Step 6: API

```php
namespace qafoo\Search;
abstract class Criterion
{
}
```

```php
namespace qafoo\Search\Criterion;
use qafoo\Search\Criterion;

class Equals extends Criterion
{
    public function __construct( $fieldName, $value )
    {}
}
```

Step 6: API

```php
namespace qafoo\Search;
abstract class Criterion
{
}
```

```php
namespace qafoo\Search\Criterion;
use qafoo\Search\Criterion;

class LogicalAnd extends Criterion
{
    public function __construct( Criterion $leftCriterion,
        Criterion $rightCriterion )
    {}
}
```

Step 6: Issues

```
1  use qafoo\Search\Search;
2
3  // Object construction
4  $search = new Search( /* ... */ );
5
6  // Search
7  $products = $search->find(
8    new LogicalAnd( /* ... */ ),
9    array(
10     'filters' => array(
11         'user' => 'john_doe'
12     )
13   )
14 );
```

Step 6: Issues

- No auto completion for arrays / strings
- Parsing potentially complex structure
- Almost impossible to extend

Keep in Mind

- Remember inversion of control
- Provide hook-in mechanism where sensible

Step 7

Step 7: Usage

```php
use qafoo\Search\Search;

// Object construction
$search = new Search( /* ... */ );

// Search
$products = $search->find(
  new LogicalAnd( /* ... */ ),
  new UserPermissionFilter( 'john_doe' )
);
```

Step 7: API

```
1  namespace qafoo\Search;
2  abstract class Filter
3  {
4      abstract public function isAccepted( Result $result );
5  }
```

Step 7: API

```php
<?php

namespace qafoo\Search\Filter;
use qafoo\Search\Filter;

class UserPermissionFilter extends Filter
{
    public function __construct( $user )
    { /* ... */ }

    public function isAccepted( Result $result )
    {
        return ( $result->creator == $this->user );
    }
}
```

Step 7: Issues

- Results are created out of thin air?
  - Results contain arbitrary objects
- Smells like . . .
  - dangerous assumptions
  - hidden dependencies
  - black magic

Keep in Mind

- Object live cycle control

Step 8

Step 8: Usage

```php
use qafoo\Search\Search;
use qafoo\Search\ResultFactory;

// Object construction
$search = new Search(
    new Backends\SolrSearchBackend( /* ... */ ),
    new Backends\ZeroMqIndexBackend( /* ... */ ),
    new ResultFactoy\ArrayResultFactory()
);

// Search
$products = $search->find(
  new LogicalAnd( /* ... */ ),
  new UserPermissionFilter( 'john_doe' )
);
```

Step 8: API

```
1  namespace qafoo\Search;
2  abstract class ResultFactory
3  {
4      abstract public function canHandle( $className );
5      abstract public function createResultObject( $className, array
           $data );
6  }
```

Step 8: API

```php
<?php

namespace qafoo\Search\ResultFactory;
use qafoo\Search\ResultFactory;

class ArrayResultFactory extends ResultFactory
{
    public function canHandle( $className )
    {
        return is_subclass_of( $className, 'ArrayObject' );
    }

    public function createResultObject( $className, array $data )
    {
        return new $className( $data );
    }
}
```

Outline

On the beauty of APIs

Getting into code

Summary

Keep in mind (random picks)

- Descriptive naming
- Know your audience
- Let API user create objects
- Never force static dependencies
- Forsee testability of using code
- Separate concerns on different levels
- Make implementations replaceable
- Remember inversion of control
- Avoid code in strings

Thanks for Listening

Stay in touch

- Tobias Schlitt
- toby@qafoo.com
- @tobySen / @qafoo

- Slides: `http://talks.qafoo.com`

Rent a web quality expert:
`http://qafoo.com`