
Designing Beautiful APIs

SFUGCGN

Tobias Schlitt (@tobySen)

May 14, 2013

About me

- ▶ Tobias Schlitt (**Toby**)

About me

- ▶ Tobias Schlitt (**Toby**)
- ▶ Degree in computer science

About me

- ▶ Tobias Schlitt (**Toby**)
- ▶ Degree in computer science
- ▶ Professional PHP since 2000

About me

- ▶ Tobias Schlitt (**Toby**)
- ▶ Degree in computer science
- ▶ Professional PHP since 2000
- ▶ Open source enthusiast

About me

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

Co-founder of



Co-founder of



**Helping teams to create high quality web application
development.**

<http://qafoo.com>

Co-founder of



Helping teams to create high quality web application development.

<http://qafoo.com>

- ▶ Expert consulting
- ▶ Individual training



<http://symfonysummit.de>

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 unforeseen 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
- ▶ beware of errors
- ▶ supports readable and testable code

So, where ...

Where do you find APIs?

Where do you find APIs?

- ▶ Web services
 - ▶ REST
 - ▶ SOAP

Where do you find APIs?

- ▶ Web services
 - ▶ REST
 - ▶ SOAP
 - ▶ Not covered here

Where do you find APIs?

- ▶ Web services
 - ▶ REST
 - ▶ SOAP
 - ▶ Not covered here
- ▶ Code
 - ▶ Libraries / frameworks
 - ▶ Layer / components / modules boundaries
 - ▶ Interfaces (and abstract classes!)

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

Levels of Beauty

- ▶ Code beauty (of the API)
 - ▶ OOP / OOD
 - ▶ Consistent, good looking code
 - ▶ Tests
- ▶ Syntactic / semantical beauty
 - ▶ Code structure
 - ▶ Naming & meaning
 - ▶ Discourage errors

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

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: 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: 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

Step 1: Issues

- ▶ Indescriptive
- ▶ Ambiguous

Keep in Mind

- ▶ Find descriptive names

Keep in Mind

- ▶ Find descriptive names
- ▶ Know your audience

Keep in Mind

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

Step 2

Step 2

Step 2: Usage

```
1 <?php
2
3 use qafoo\Search\Search;
4
5 // Index
6
7 $product = new Product( 'Glow_Stone_Driveway' );
8 Search::getInstance()->index( $product );
9
10 // Search
11
12 $products = Search::getInstance()->find(
13     'type_="Product" _AND_name_LIKE_"%stone%"
14 );
```

Step 2: Usage

```
1 <?php
2
3 use qafoo\Search\Search;
4
5 // Index
6
7 $product = new Product( 'Glow_Stone_Driveway' );
8 Search::getInstance()->index( $product );
9
10 // Search
11
12 $products = Search::getInstance()->find(
13     'type_="Product" _AND_name_LIKE_"%stone%"
14 );
```

Step 2: Usage

```
1 <?php
2
3 use qafoo\Search\Search;
4
5 // Index
6
7 $product = new Product( 'Glow_Stone_Driveway' );
8 Search::getInstance()->index( $product );
9
10 // Search
11
12 $products = Search::getInstance()->find (
13     'type_=_"Product" _AND_name_LIKE_"%stone%"
14 );
```

Step 2: API

```
1 namespace qafoo\Search;
2 class Search
3 {
4     public static function getInstance()
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 2: Issues

- ▶ Not configurable

Step 2: Issues

- ▶ Not configurable
- ▶ Not easily mockable

Step 2: Issues

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

Keep in Mind

- ▶ Don't force to violate Law of Demeter (LoD)

Keep in Mind

- ▶ Don't force to violate Law of Demeter (LoD)
- ▶ Leave instancitation / configuration to the user

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

Step 3: Usage

```
1 <?php
2
3 use qafoo\Search\Search;
4
5 // Index
6
7 $search = new Search();
8 $product = new Product( 'Glow_Stone_Driveway' );
9 $search->index( $product );
10
11 // Search
12
13 $search = new Search();
14 $products = $search->find(
15     'type_="Product" _AND_name_LIKE_"%stone%"
16 );
```

Step 3: Usage

```
1 <?php
2
3 use qafoo\Search\Search;
4
5 // Index
6
7 $search = new Search();
8 $product = new Product( 'Glow_Stone_Driveway' );
9 $search->index( $product );
10
11 // Search
12
13 $search = new Search();
14 $products = $search->find(
15     'type_="Product" _AND_name_LIKE_"%stone%"
16 );
```


Step 3: Usage

```
1 <?php
2
3 use qafoo\Search\Search;
4
5 // Index
6
7 $search = new Search();
8 $product = new Product( 'Glow_Stone_Driveway' );
9 $search->index( $product );
10
11 // Search
12
13 $search = new Search();
14 $products = $search->find(
15     'type = "Product" _AND_ name _LIKE_ "%stone%" '
16 );
```

Step 3: API

```
1 namespace qafoo\Search;
2 class Search
3 {
4     public function __construct()
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 3: Issues

- ▶ Search & index different concerns?

Step 3: Issues

- ▶ Search & index different concerns?
- ▶ Not easily replaceable

Step 3: Issues

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

Keep in Mind

- ▶ Make implementations replaceable

Keep in Mind

- ▶ Make implementations replaceable
- ▶ Stay flexible for changing requirements

Keep in Mind

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

Step 4

Step 4

Step 4: Usage

```
1 use qafoo\Search\Search;
2
3 // Object construction, should take place in DIC!
4 $search = new Search(
5     new Backends\SolrSearchBackend( /* ... */ ),
6     new Backends\ZeroMqIndexBackend( /* ... */ )
7 );
8
9 // Index
10 $product = new Product( 'Glow_Stone_Driveway' );
11 $search->index( $product );
12
13 // Search
14 $products = $search->find(
15     'type = "Product" _AND_ name _LIKE_ "%stone%" '
16 );
```

Step 4: Usage

```
1 use qafoo\Search\Search;  
2  
3 // Object construction, should take place in DIC!  
4 $search = new Search(  
5     new Backends\SolrSearchBackend( /* ... */ ),  
6     new Backends\ZeroMqIndexBackend( /* ... */ )  
7 );  
8  
9 // Index  
10 $product = new Product( 'Glow_Stone_Driveway' );  
11 $search->index( $product );  
12  
13 // Search  
14 $products = $search->find(  
15     'type_="Product" _AND_name_LIKE_"%stone%"'  
16 );
```

Step 4: Usage

```
1 use qafoo\Search\Search;
2
3 // Object construction, should take place in DIC!
4 $search = new Search(
5     new Backends\SolrSearchBackend( /* ... */ ),
6     new Backends\ZeroMqIndexBackend( /* ... */ )
7 );
8
9 // Index
10 $product = new Product( 'Glow_Stone_Driveway' );
11 $search->index( $product );
12
13 // Search
14 $products = $search->find(
15     'type_="Product" _AND_name_LIKE_"%stone%"
16 );
```

Step 4: Usage

```
1 use qafoo\Search\Search;
2
3 // Object construction, should take place in DIC!
4 $search = new Search(
5     new Backends\SolrSearchBackend( /* ... */ ),
6     new Backends\ZeroMqIndexBackend( /* ... */ )
7 );
8
9 // Index
10 $product = new Product( 'Glow_Stone_Driveway' );
11 $search->index( $product );
12
13 // Search
14 $products = $search->find(
15     'type = "Product" _AND_ name _LIKE_ "%stone%" '
16 );
```

Step 4: API

```
1 namespace qafoo\Search;
2 class Search
3 {
4     public function __construct( SearchBackend $searchBackend,
5         IndexBackend $indexBackend )
6     {}
7     public function find( $searchQuery, array $config = array() )
8     {}
9     public function suggest( $prefix, array $fields )
10    {}
11    public function index( $object )
12    {}
13 }
```

Step 4: Issues

- ▶ Back ends use different syntax

Step 4: Issues

- ▶ Back ends use different syntax
- ▶ Writing strings in code sucks

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

Keep in Mind

- ▶ Avoid code is strings
- ▶ Separate concerns

Step 5

Step 5

Step 5: Usage

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

Step 5: Usage

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

Step 5: Usage

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

Step 5: Usage

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


Step 5: Usage

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

Step 5: Usage

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

Step 5: API

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

Step 5: API

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

Step 5: API

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

Step 5: API

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

Step 5: API

```
1 namespace qafoo\Search;
2 class Criterion
3 {
4     const EQUALS = '=';
5     const LIKE = 'LIKE';
6     const LAND = '&&';
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: API

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

Step 5: Issues

- ▶ Criteria hardly extensible

Step 5: Issues

- ▶ Criteria hardly extensible
- ▶ Parent ctor should always be valid

Keep in Mind

- ▶ Never use constants for processing instructions

Keep in Mind

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

Step 6

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: 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: 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: 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

```
1 namespace qafoo\Search;  
2 abstract class Criterion  
3 {  
4 }
```

```
1 namespace qafoo\Search\Criterion;  
2 use qafoo\Search\Criterion;  
3  
4 class Equals extends Criterion  
5 {  
6     public function __construct( $fieldName , $value )  
7     {}  
8 }
```

Step 6: API

```
1 namespace qafoo\Search;  
2 abstract class Criterion  
3 {  
4 }
```

```
1 namespace qafoo\Search\Criterion;  
2 use qafoo\Search\Criterion;  
3  
4 class Equals extends Criterion  
5 {  
6     public function __construct( $fieldName, $value )  
7     {}  
8 }
```

Step 6: API

```
1 namespace qafoo\Search;  
2 abstract class Criterion  
3 {  
4 }
```

```
1 namespace qafoo\Search\Criterion;  
2 use qafoo\Search\Criterion;  
3  
4 class Equals extends Criterion  
5 {  
6 public function __construct( $fieldName, $value )  
7     {}  
8 }
```

Step 6: API

```
1 namespace qafoo\Search;  
2 abstract class Criterion  
3 {  
4 }
```

```
1 namespace qafoo\Search\Criterion;  
2 use qafoo\Search\Criterion;  
3  
4 class LogicalAnd extends Criterion  
5 {  
6     public function __construct( Criterion $leftCriterion ,  
7         Criterion $rightCriterion )  
8     {}  
9 }
```

Step 6: API

```
1 namespace qafoo\Search;  
2 abstract class Criterion  
3 {  
4 }
```

```
1 namespace qafoo\Search\Criterion;  
2 use qafoo\Search\Criterion;  
3  
4 class LogicalAnd extends Criterion  
5 {  
6     public function __construct( Criterion $leftCriterion ,  
7         Criterion $rightCriterion )  
8     {}  
9 }
```

Step 6: API

```
1 namespace qafoo\Search;  
2 abstract class Criterion  
3 {  
4 }
```

```
1 namespace qafoo\Search\Criterion;  
2 use qafoo\Search\Criterion;  
3  
4 class LogicalAnd extends Criterion  
5 {  
6     public function __construct( Criterion $leftCriterion ,  
7         Criterion $rightCriterion )  
8 }
```


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

Step 6: Issues

- ▶ No auto completion for arrays / strings
- ▶ Parsing potentially complex structure

Step 6: Issues

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

Keep in Mind

- ▶ Remember inversion of control

Keep in Mind

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

Step 7

Step 7

Step 7: Usage

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


Step 7: Usage

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

Step 7: API

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

Step 7: API

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

Step 7: API

```
1 <?php
2
3 namespace qafoo\Search\Filter;
4 use qafoo\Search\Filter;
5
6 class UserPermissionFilter extends Filter
7 {
8     public function __construct( $user )
9     { /* ... */ }
10
11     public function isAccepted( Result $result )
12     {
13         return ( $result->creator == $this->user );
14     }
15 }
```

Step 7: API

```
1 <?php
2
3 namespace qafoo\Search\Filter;
4 use qafoo\Search\Filter;
5
6 class UserPermissionFilter extends Filter
7 {
8     public function __construct( $user )
9     { /* ... */ }
10
11     public function isAccepted( Result $result )
12     {
13         return ( $result->creator == $this->user );
14     }
15 }
```

Step 7: API

```
1 <?php
2
3 namespace qafoo\Search\Filter;
4 use qafoo\Search\Filter;
5
6 class UserPermissionFilter extends Filter
7 {
8     public function __construct( $user )
9     { /* ... */ }
10
11     public function isAccepted( Result $result )
12     {
13         return ( $result->creator == $this->user );
14     }
15 }
```

Step 7: Issues

- ▶ Results are created out of thin air?
 - ▶ Results contain arbitrary objects

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

Step 8: Usage

```
1 use qafoo\Search\Search;
2 use qafoo\Search\ResultFactory;
3
4 // Object construction
5 $search = new Search(
6     new Backends\SolrSearchBackend( /* ... */ ),
7     new Backends\ZeroMqIndexBackend( /* ... */ ),
8     new ResultFactory\ArrayResultFactory()
9 );
10
11 // Search
12 $products = $search->find(
13     new LogicalAnd( /* ... */ ),
14     new UserPermissionFilter( 'john_doe' )
15 );
```

Step 8: Usage

```
1 use qafoo\Search\Search;  
2 use qafoo\Search\ResultFactory;  
3  
4 // Object construction  
5 $search = new Search(  
6     new Backends\SolrSearchBackend( /* ... */ ),  
7     new Backends\ZeroMqIndexBackend( /* ... */ ),  
8     new ResultFactory\ArrayResultFactory()  
9 );  
10  
11 // Search  
12 $products = $search->find(  
13     new LogicalAnd( /* ... */ ),  
14     new UserPermissionFilter( 'john_doe' )  
15 );
```

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  
6         $data );  
7 }
```

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  
6         $data );  
7 }
```

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  
6         $data );  
7 }
```

Step 8: API

```
1 <?php
2
3 namespace qafoo\Search\ResultFactory;
4 use qafoo\Search\ResultFactory;
5
6 class ArrayResultFactory extends ResultFactory
7 {
8     public function canHandle( $className )
9     {
10         return is_subclass_of( $className, 'ArrayObject' );
11     }
12
13     public function createResultObject( $className, array $data )
14     {
15         return new $className( $data );
16     }
17 }
```


Step 8: API

```
1 <?php
2
3 namespace qafoo\Search\ResultFactory;
4 use qafoo\Search\ResultFactory;
5
6 class ArrayResultFactory extends ResultFactory
7 {
8     public function canHandle( $className )
9     {
10         return is_subclass_of( $className, 'ArrayObject' );
11     }
12
13     public function createResultObject( $className, array $data )
14     {
15         return new $className( $data );
16     }
17 }
```

Step 8: API

```
1 <?php
2
3 namespace qafoo\Search\ResultFactory;
4 use qafoo\Search\ResultFactory;
5
6 class ArrayResultFactory extends ResultFactory
7 {
8     public function canHandle( $className )
9     {
10         return is_subclass_of( $className, 'ArrayObject' );
11     }
12
13     public function createResultObject( $className, array $data )
14     {
15         return new $className( $data );
16     }
17 }
```

Outline

On the beauty of APIs

Getting into code

Summary

Keep in mind (random picks)

- ▶ Descriptive naming
- ▶ Know your audience

Keep in mind (random picks)

- ▶ Descriptive naming
- ▶ Know your audience
- ▶ **Let API user create objects**

Keep in mind (random picks)

- ▶ Descriptive naming
- ▶ Know your audience
- ▶ Let API user create objects
- ▶ Never force static dependencies
- ▶ Forsee testability of using code

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**

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

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**

Questions?

Stay in touch

- ▶ Tobias Schlitt
- ▶ toby@qafoo.com
- ▶ @tobySen / @qafoo
- ▶ Slides: <http://talks.qafoo.com>

Rent a web quality expert:
<http://qafoo.com>

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>