The Insider's Guide to CakePHP Interviewing

Fully mastering CakePHP can take some time, which makes finding true CakePHP experts a real challenge. The questions presented in this guide can be highly effective in evaluating the breadth and depth of a developer's knowledge of the CakePHP framework.

HIRE A TOP CAKEPHP DEVELOPER NOW

2.6k Share 12 12

CakePHP is an extensive framework whose feature set has continued to grow substantially since its initial release in 2005. As a result, fully mastering its capabilities can take some time, which makes finding true CakePHP experts a real challenge.

Finding them requires a highly-effective recruiting process, as described in our post on Finding and Hiring the Best in the Industry. Such a process can then be augmented with questions – such as those presented herein – to identify those candidates who have truly mastered CakePHP.

Q: How do CakePHP conventions and Object-Relational Mapping (ORM) help streamline queries? Also discuss any potential pitfalls.

CakePHP’s Object-relational mapping (ORM) benefits greatly from CakePHP conventions. By setting out the database schema to Cake’s standards, you can quickly connect tables together through Cake’s powerful ORM. You rarely need to write an SQL statement, as CakePHP handles things like table joins, `a Ma`, and even `a A dBe     T Ma` relationships with ease.

Leveraging CakePHP’s `C   a  ab eBe a`, through your model associations you can specify which database tables and fields to select from an SQL query. This can go several tables deep, and through the ORM it is easy to rapidly construct highly complex SQL statements. It helps you search and filter data in a clean and consistent way and can also help increase the speed and overall performance of your application. (It works by temporarily or permanently altering the associations of your models, using the supplied containments to generate a corresponding series of `b  dM de` and `b  dM de` calls.)

Overall, Cake’s ORM really does help streamline development and, if used correctly, is an amazing tool for building complex queries quickly. It is nonetheless vital that developers take the time to fully understand the ORM and to ensure that their queries are are properly optimized (as is true in any language).

The challenge with the ORM is that it makes using SQL so simple that, if a developer isn’t careful, he or she can write inefficient SQL queries without meaning to. These problems tend to surface after a system has been deployed, as databases grow and badly written queries become increasingly slow.

Q: What are CakePHP Helpers? List the 10 types of Helpers that are available and, using the `FormHelper` as an example, describe how Helpers can be used to speed up development.

CakePHP Helpers are component-like classes for the presentation layer of your application. They contain logic that can be shared by many views, elements, or layouts.

The 10 types of helpers available in CakePHP are:

1. CacheHelper
2. FormHelper
3. HtmlHelper
4. JsHelper
5. NumberHelper
6. Paginator
7. RSS
8. SessionHelper
9. TextHelper
10. TimeHelper

By encapsulating commonly used functionality in reusable form, CakePHP Helpers help speed up development. A great example is the FormHelper, which creates your form input fields based on your database table schema set up. For example, a TINY INT field will automatically be mapped to a checkbox, while a TEXT field will automatically be mapped to a text area.

By ensuring that the FormHelper names match the same names as in the database table, the form will automatically be created.

Using your data validation rules, CakePHP will *automatically* display error messages next to the form input if the data validation fails. All that is required is for the developer to match the form input to the fields in the database. With a little bit of help from the controller, the data will then automatically be saved to the database once the validation has passed successfully.

Q: What are Components and what are the benefits of using them? Provide an example of how you would access a component via a controller.

Components are logical modules that are shared between controllers. CakePHP comes with its own set of core components for common tasks and you can also create your own components. Creating and using components can help keep controller code clean and facilitates code reuse across and between projects. For example, you might want to create a custom “shopping cart” component for use across multiple controllers in an e-commerce application.

Each component you include in a controller is exposed as a property on that controller. For example, if you included the `Se      C      e` and the `C     eC       e` in your controller, you could access them as follows:

```
c a   P    C        e  e  e d  A  C          e
      b  c $c      e     = a  a ('Se       ', 'C    e');

      b  c    c      de e e()
           ($     ->P   ->de e e($     -> e  e  ->da a('P    d'))
            $     ->Se      -> e F a  ('P    de e ed.');
             e     $     -> ed  ec (a  a ('ac     ' => '  de '));
```

Q: What are Behaviors and what are their advantages? List the 4 Behaviors supported “out of the box” in CakePHP.

In much the same way that a Component extends a Controller, a Behavior extends a Model. Behaviors enable you to separate and reuse logic that performs a type of behavior, and to do so without requiring inheritance.

As an example, consider a model that provides access to a database table which stores structural information about a tree. Removing, adding, and migrating nodes in the tree is not as simple as deleting, inserting, and editing rows in the table. Rather than creating those tree-manipulation methods on a per model basis (for every model that needs that functionality), we could simply attach the TreeBehavior to our model.

The following 4 Behaviors are provided "out of the box" in CakePHP:

1. AclBehavior: provides a way to seamlessly integrate a model with your ACL (Access Control List) system
2. ContainableBehavior: streamlines search and filter operations
3. TranslateBehavior: for internationalization
4. TreeBehavior: facilitates accessing and manipulating hierarchical data in database tables

You can also create your own behaviors. Behaviors can be a great way to keep a clean code base and keep your code out of your controller. A good example is the open source `I a eU   adBe a` which allows for a very simple image or file uploading. By specifying a few rules in your model file, files will be automatically validated and uploaded, with no extra code required in your Controller file. An added plus of that particular behavior is that it uses PHP Thumb to automatically resize any image uploads.

Q: How would you handle nested data in CakePHP, such as a category tree?

Creating a category structure which goes unlimited levels deep is a good example of where the CakePHP's TreeBehavior can come in handy.

Employing the TreeBehavior is trivially simple and is done as follows in your model file:

```
  b  c $ac   A  = 'T ee';
```

The categories table might then look something like this:

```
CREATE TABLE ca e    e  (
     d INTEGER(10) UNSIGNED NOT NULL AUTO_INCREMENT,
     a e  _ d INTEGER(10) DEFAULT NULL,
        INTEGER(10) DEFAULT NULL,
         INTEGER(10) DEFAULT NULL,
     a e VARCHAR(255) DEFAULT '',
    PRIMARY KEY  ( d)
);
```

By setting the `a e  _ d` field when you save the data, the and fields will automatically be populated. The and fields follow an MPTT (Modified Preorder Tree Traversal) structure.

All of the TreeBehavior's methods are then available for use. Examples include:

- `e e a eT eeL   ()` - returns a hierarchical array of values (e.g., for use with HTML select boxes, etc.)
- `c   d e ($ a e  Id)` - returns a list of the children of the specified `$ a e  Id`
- `e Pa   ($ d)` - returns an array of the nodes to traverse hierarchically within the tree to reach the specified `$ d`

Q: How do you perform data validation in CakePHP?

CakePHP simplifies data validation, enabling you to specify in your model file the data validation rules for each of your database tables that are universal for that model. This adheres to the DRY

(Don’t Repeat Yourself) principle by enabling you to just specify the rules once and then have them apply across the entire model.

The supported validation rules are plentiful in CakePHP. Here are a few good examples:

```
  b  c $ a  da e = a  a (
        '    e' => a  a (
                '   e' => '    e',
                ' e  a e' => 'A  a  d     e    be      e    ed.'
        ),
        'e a  ' => a  a (
                '   e' => 'e a  ',
                ' e  a e' => 'A  a  d e a   add e       e    ed.'
        ),
        ' a     d' => a  a (
                '   e'    => a  a ('   Le    ', '8'),
                ' e  a e' => 'M       8 c a ac e        '
        ),
        'd b' => a  a (
                '   e'    => 'da e',
                ' e  a e' => 'E  e  a  a  d da e',
                'a    E    ' =>     e
        )
);
```

Note in particular the `a     E` key in the `'d b'` array which allows the field to be empty. Based on these validation rules, the dob field can be left blank, but if a date is entered, it will be checked to confirm that it is a valid date value.

Q: Provide some examples of folder and file manipulation in CakePHP.

Often challenging with standard PHP alone, the Folder & File Utilities are useful if you need to create, upload, or manipulate folders or files.

Here are some key examples:

The `F  de ::c` method simplifies copying a file from one location to another:

```
// C       de 1 a d a       c    e            de 2
$   de 1 =  e  F  de ('/ a  /  /   de 1');
$   de 1->c   ('/ a  /  /   de 2');
```

The `F  de ::c` method supports additional options as well. Specifically:

```
$   de  =  e  F  de ('/ a  /  /   de ');
$   de ->c   (a  a (
    '  ' => '/ a  /  / e /   de ',
    '    ' => '/ a  /  /c   /     ',
    '  de' => 0755,
    '    ' => a  a ('    - e.    ', '.    '),
    ' c e e' => F  de ::SKIP  // S     d  ec    e /   e     a  a  ead  e
));
```

Or to create new folders, simply use the `F  de ::c ea e` method:

```
$   de  =  e  F  de ();
   ($   de ->c ea e('    ' . DS . 'ba ' . DS . 'ba ' . DS . '    e' . DS . '     '))
    // S cce        c ea ed   e  e  ed    de
```

The `F  de ::    d` method is particularly useful as it enables you to dynamically find files within a directory:

```
// F  d a   .            a  / eb    /   /    de  a d        e  e
$d   =  e  F  de (WWW_ROOT . '   ');
$   e  = $d  ->   d('.*\.   ',    e);
```

Q: What are some advantages of the "Fat Model, Skinny Controller" approach? Provide an example of how you would use it in CakePHP.

"Fat Model, Skinny Controller" - often advocated by Ruby on Rails developers - is an approach within the Model/View/Controller (MVC) architectural paradigm whereby logic should predominantly exist within the model. This relegates the "skinny" controller to its intended role as a controlling interface between the view and model.

Consider, for example, performing a simple CRUD (create, read, update and delete) operation, such as adding posts to a blog. The default add method might be as follows:

```
b  c    c     add()
     ($    -> e  e  ->  ('    '))
      $    ->P   ->c ea e();
         ($    ->P   -> a e($    -> e  e  ->da a))
          $    ->Se     -> e F a  (__('Y        a  bee   a ed.'));
           e     $    -> ed  ec (a  a ('ac    ' => '  de '));

      $    ->Se     -> e F a  (__('U ab e   add          .'));
```

This controller action is fine for a simple add, but what would happen if you wanted to do things such as send an email to the admin when a post was added, or update another model association when a post was added? This is additional logic, but this logic shouldn't go in the controller file.

Instead we would write a method for this in our `P    .` model, perhaps something like this:

```
b  c    c     addP   ($da a = a  a (), $e a  Ad    =    e)
   $    ->c ea e();
   $    -> a e($da a);

   //   da e a      e   ab e

     ($e a  Ad   )
      //  e d   e e a        e ad     e


   //    a         cce
    e        e;
```

This would then only require a small change to the controller action as follows:

```
b  c    c     add()
     ($    -> e  e  ->  ('    '))
         ($    ->P   ->addP   ($    -> e  e  ->da a))
          $    ->Se     -> e F a  (__('Y        a  bee   a ed.'));
           e     $    -> ed  ec (a  a ('ac    ' => '  de '));

      $    ->Se     -> e F a  (__('U ab e   add          .'));
```

As you can see, the new action is actually one less line, because the `$    ->P   ->c ea e()` has been moved to the model file, helping achieve clean and concise code.

Q: How does CakePHP handle authentication and user login? Provide a code example.

CakePHP has a built in authentication component (AuthComponent) that makes setting up a user registration and login system very straightforward. By setting up a users table with a username or email field and a password field, a programmer can quickly incorporate authentication into their application. CakePHP also handles password encryption, providing several different classes for encryption including B crypt and digest authentication. CakePHP also has some advanced methods for doing things such as logging users in automatically.

Here’s how you can easily take advantage of these capabilities in your code:

Your database table might look something like this:

```
CREATE TABLE    e   (
     d INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
      e  a e VARCHAR(50),
     a     d VARCHAR(255),
       e VARCHAR(20),
    c ea ed DATETIME DEFAULT NULL,
      d   ed DATETIME DEFAULT NULL
);
```

In your `A  C        e .` you would then setup the component:

```
// a  /C       e /A  C       e .
c a   A  C       e  e  e d  C       e

      b  c $c     e    = a  a (
        'Se     ',
        'A   ' => a  a (
            '     Red  ec ' => a  a (
                'c       e ' => '     ',
                'ac     ' => '  de'
            ),
            '      Red  ec ' => a  a (
                'c       e ' => ' a e',
                'ac     ' => 'd    a',
                '   e'
            ),
            'a    e   ca e' => a  a (
                'F   ' => a  a (
                    ' a     dHa  e   => 'B        '
                )
            )
        )
    );
```

In your `U e .` model file, you would set the password encryption code in the `be   eSa e` callback:

```
// a  /M de /U e .

A  ::  e ('A  M de ', 'M de ');
A  ::  e ('B       Pa     dHa  e ', 'C       e /C     e  /A   ');

c a   U e  e  e d  A  M de

// ...

  b  c    c     be   eSa e($        = a  a ())
     (   e ($    ->da a[$    ->a  a ][' a     d']))
      $ a     dHa  e  =  e  B       Pa     dHa  e ();
      $    ->da a[$    ->a  a ][' a     d'] = $ a     dHa  e -> a  (
          $    ->da a[$    ->a  a ][' a     d']
      );

     e        e;
```

Then, in your `U e  C        e .` you can set the login action:

```
// a  /C       e /U e  C       e .

  b  c    c           ()
       ($     -> e  e  ->  ('  ,  '))
           ($     ->A    ->       ())
             e     $     -> ed  ec ($    ->A    -> ed  ec ());

        $     ->Se      -> e F a   (__('I  a  d   e a e     a     d,     a a  '));
```

Note how simple it is. The code simply looks for a post, and then calls the `A     ->` component method which logs the user in.

The view login file might then look something like this:

```
//a  /V e /U e  /     .c

<d   c a  ="  e       ">
<?    ec   $     ->F   ->c ea e('U e ');?>
    <  e d e >
        < e e d>
            <?    ec   __('P ea e e  e          e  a e a d  a     d'); ?>
        </ e e d>
        <?    ec   $     ->F    ->     ('  e  a e');
        ec   $     ->F    ->     (' a     d');
    ?>
    </  e d e >
<?    ec   $     ->F   ->e d(__('L    ')); ?>
</d  >
```

Just set the form with the username and password fields, and CakePHP handles the rest.

Finally, for the logout script, in your `U e  C        e` the logout action could look something like this:

```
  b  c    c           ()
     e     $     -> ed  ec ($     ->A    ->      ());
```

Q: Provide an example of how you would use CakePHP's callbacks.

CakePHP callbacks enable you to manipulate or check data before a model operation. Examples include before validation, before save, after save, before delete, after delete, and after find.

For example, consider a case where you want to manipulate a date so that it is displayed differently than the way it is saved in the database. (Perhaps you are working with an older database that saved the date in a `    e()` format or in a non standard database date format.)

To accomplish this, these callbacks could go in your model file:

```
  b  c    c     a  e F  d($ e     , $    a    =  a  e)
       eac  ($ e      a  $ e  => $ a )
           (   e ($ a ['E e  ']['be   da e']))
             $ e     [$ e ]['E e  ']['be   da e'] = $     ->da eF   a A  e F  d(
                 $ a ['E e  ']['be   da e']
             );


     e     $ e     ;


  b  c    c     da eF   a A  e F  d($da eS     )
     e     da e('d- -Y',         e($da eS     ));
```

The `a  e F  d` callback will take the data returned from a find query, and change the format of the date. In this example, we set the date to be in the `d- -Y` format – perhaps preparing the data for going into a date picker or something similar. This code will be called before the data is returned to the controller, so it allows us to manipulate data before we receive it in our controller.

We would also need to have a callback before we save the data, to revert the date format back to that used in the database:

```
  b  c    c     be   eSa e($        = a  a ())
      (!e    ($    ->da a['E e  ']['be   da e']))
       $    ->da a['E e  ']['be   da e'] = $    ->da eF   a Be   eSa e(
           $    ->da a['E e  ']['be   da e']
       );

       // Be    e     e        e,         a e              a  !
   e        e;


  b  c    c     da eF   a Be   eSa e($da eS      )
   e     da e('Y- -d',         e($da e$     ));
```

Q: What are Virtual Fields in CakePHP? How and why would you use them. Provide an example.

Virtual Fields allow you to create arbitrary SQL expressions and assign them as fields in a model. These fields cannot be saved, but will be treated like other model fields for read operations. They will be indexed under the model's key alongside other model fields.

As a simple example, consider a model that contains `     _ a e` and `a   _ a e` fields. You might find that you often want to use the user's full name. In this case, in your model file you can add:

```
  b  c $     a F e d  = a  a (
    '    _ a e' => 'CONCAT(U e .     _ a e, " ", U e . a  _ a e)'
);
```

This will add a new field called `     _ a e`. When doing a find query, the data would then show as follows:

```
A  a
(
    [0] => A  a
        (
            [U e ] => A  a
                (
                    [ d] => 1
                    [    _ a e] => J
                    [ a  _ a e] => S
                    [    _ a e] => J    S
                )
        )
)
```

Another good use of virtual fields is when you need to count data. Let's take the example when an article has comments. Often we need to count how many comments an article has. This is simple to do with virtual fields. For example, in your model file, you could add the following:

```
  b  c $     a F e d  = a  a (
        '   _c    e    ' => 'SELECT COUNT( d) FROM c    e    WHERE a    c e_ d = A    c e. d'
);
```

This will add `   _c    e` at the end of any Article find query, e.g:

```
A  a
(
    [0] => A  a
        (
            [A    c e] => A  a
                (
                    [ d] => 1
                    [    e] => Te   A   c e
                    [de c       ] => Te   De c
                    [   _c   e   ] => 2
                )
        )
)
```

A c AnHn A fc n m n b H

Virtual fields do not come without any penalty though. Their downside is performance. This should be kept in mind when creating virtual fields, as the more complex your virtual field, the more impact on query performance the virtual field will have.

e

Wrap-up

The questions presented in this guide can be highly effective in evaluating the breadth and depth of a developer’s knowledge of the CakePHP framework. It is important to bear in mind, though, that these questions are intended merely as a guide. Not every “A” candidate worth hiring will be able to properly answer them all, nor does answering them all guarantee an “A” candidate. At the end of the day, hiring remains as much of an art as it does a science.

See also: Toptal's growing, community-driven list of great CakePHP interview questions.

Recent Cak e ca ee b

Join the Toptal community.

HIRE A DEVELOPER

OR

APPLY AS A DEVELOPER

TRENDING ON BLOG

- Why Should I Learn Scala?
- AngularJS: Demystifying Directives
- Simple Data Flow in React Applications Using Flux and Backbone
- From Objective-C to Swift
- The 5 Most Common HTML5 Mistakes
- 3D Data Visualization with Open Source Tools: An Example Using VTK

NAVIGATION

- Why
- How
- What
- Clients
- Team
- Community
- Blog
- Resources
- Client reviews
- Developer reviews
- Contact
- FAQ

CONTACT

- Apply for work
- Become a partner
- Send us an email
- Call 888.604.3188

SOCIAL

- Facebook
- Twitter
- Google+
- GitHub
- Dribbble

Exclusive access to top developers