

2026 CakePHP-Day5

CakePHP CMS 教學3

認證功能

現在我們的 CMS 已經有了使用者，我們可以使用 `cakephp/authentication` 外掛為他們啟用登入功能。我們將從確保密碼在資料庫中被安全地儲存開始。接著，我們會實現登入與登出功能，並讓新使用者可以註冊帳號。

安裝認證外掛

- 使用 Composer 安裝 Authentication 外掛：

```
composer require "cakephp/authentication:^4.0"
```

```
bash
```

認證功能

新增密碼雜湊

- 在開始之前，你需要為資料庫中的 `users` 資料表建立對應的 `Controller`、`Table`、`Entity` 和模板。你可以手動建立，就像之前為 `ArticlesController` 所做的，或者使用 `bake` 指令來自動生成相關類別：

```
bin/cake bake all users
```

```
bash
```

- 當你使用此設置來新增或更新使用者時，可能會發現密碼以純文字形式儲存。從安全角度來看，這是非常危險的，因此我們需要修正這個問題。
- 這也是討論 `CakePHP` 中模型層的好時機。在 `CakePHP` 中，我們使用不同的類別來操作記錄集合和單一記錄。在 `CakePHP` 中，`Table` 類別主要負責操作一組資料紀錄，而 `Entity` 則代表單筆資料。因為密碼雜湊是「單一使用者資料」的行為，所以我們會將這段邏輯放在 `User Entity` 中。
- 例如，密碼的雜湊處理是針對單一記錄的，因此我們會在 `Entity` 中實現此行為。因為我們希望每次設定密碼時都進行雜湊處理，所以會使用一個修改器/設置器方法。`CakePHP` 會在每次設定實體的屬性時，自動調用基於約定的 `setter` 方法。

認證功能

在 src/Model/Entity/User.php 中新增以下程式碼：

// 檔案：src/Model/Entity/User.php

User.php

```
<?php
namespace App\Model\Entity;

use Authentication\PasswordHasher\DefaultPasswordHasher; // 新增此行
use Cake\ORM\Entity;

class User extends Entity
{
    // Bake 生成的程式碼。

    // 新增以下方法
    protected function _setPassword(string $password): ?string
    {
        if (mb_strlen($password) > 0) {
            return (new DefaultPasswordHasher())->hash($password);
        }
        return null;
    }
}
```

認證功能

測試與驗證

- 在瀏覽器中訪問 `http://localhost:8080/users`，即可看到使用者列表。
- 你可以編輯安裝時創建的預設使用者。如果更改該使用者的密碼，應該會在列表或檢視頁面中看到一個經過雜湊處理的密碼，而不是原始值。
- CakePHP 預設使用 `bcrypt` 來進行密碼雜湊。我們建議所有新應用程式都使用 `bcrypt`，以保持高安全標準。這是 PHP 推薦的密碼雜湊算法。
- 現在為至少一個使用者帳號創建一個雜湊密碼！這將在下一步中使用。當你更新密碼後，應該會在 `password` 欄位中看到一個很長的字串。請注意，即使是相同的密碼，每次儲存時 `bcrypt` 都會生成不同的雜湊值。

新增登入功能

現在是時候來配置認證外掛了。該外掛將透過三個不同的類別來處理認證過程：

1. Application

Application 將使用 **AuthenticationMiddleware** 並提供一個 **AuthenticationService**，該服務負責配置我們希望如何檢查憑證以及從哪裡獲取它們。

2. AuthenticationService

這是一個實用類別，允許你配置認證過程。

3. AuthenticationMiddleware

這會作為中介軟體隊列的一部分被執行，在框架處理 **Controller** 之前，它會提取憑證並處理以檢查使用者是否已認證。

之前我們使用 **AuthComponent** 處理這些步驟，現在邏輯被分為特定的類別，認證過程發生在 **Controller** 層之前。它首先根據你提供的配置檢查使用者是否已認證，然後將使用者和認證結果注入請求中，供後續參考。

新增登入功能

在 src/Application.php 中新增以下匯入：

```
use Authentication\AuthenticationService;  
use Authentication\AuthenticationServiceInterface;  
use Authentication\AuthenticationServiceProviderInterface;  
use Authentication\Identifier\PasswordIdentifier;  
use Authentication\Middleware\AuthenticationMiddleware;  
use Psr\Http\Message\ServerRequestInterface;
```

Application.php

新增登入功能

接著，在應用程式的已實現介面中，新增 `AuthorizationServiceProviderInterface`：

```
// 檔案: src/Application.php
class Application extends BaseApplication
    implements AuthenticationServiceProviderInterface
{
```

Application.php

新增登入功能

新增以下方法：

```
// 檔案: src/Application.php
public function middleware(MiddlewareQueue $middlewareQueue): MiddlewareQueue
{
    $middlewareQueue
        // ... 其他已新增的中介軟體
        ->add(new RoutingMiddleware($this))
        ->add(new BodyParserMiddleware())
        // 新增 AuthenticationMiddleware。它應位於路由與 Body Parser 後。
        ->add(new AuthenticationMiddleware($this));

    return $middlewareQueue;
}
```

Application.php

新增登入功能

新增以下方法：

Application.php

```
// src/Application.php
public function getAuthenticationService(ServerRequestInterface $request): AuthenticationServiceInterface
{
    $service = new AuthenticationService();

    // 定義使用者未通過認證時應重新導向的位置。
    $service->setConfig([
        'unauthenticatedRedirect' => [
            'prefix' => false,
            'plugin' => null,
            'controller' => 'Users',
            'action' => 'login',
        ],
        'queryParams' => 'redirect',
    ]);

    $fields = [
        PasswordIdentifier::CREDENTIAL_USERNAME => 'email',
        PasswordIdentifier::CREDENTIAL_PASSWORD => 'password',
    ];

    // 後面還有...
}
```

新增登入功能

新增以下方法：

Application.php

```
// src/Application.php
public function getAuthenticationService(ServerRequestInterface $request): AuthenticationServiceInterface
{
    // 前面剛才輸入過了...

    // 載入認證器 (Authenticators) ，並且應優先載入 Session 認證器。
    $service->loadAuthenticator('Authentication.Session');
    $service->loadAuthenticator('Authentication.Form', [
        'fields' => $fields,
        'loginUrl' => [
            'prefix' => false,
            'plugin' => null,
            'controller' => 'Users',
            'action' => 'login',
        ],
        'identifier' => [
            'className' => 'Authentication.Password',
            'fields' => $fields,
        ],
    ]);

    return $service;
}
```

新增登入功能

在 ApplicationController 中新增以下程式碼：

```
// 檔案: src/Controller/AppController.php
```

AppController.php

```
public function initialize(): void
```

```
{
```

```
    parent::initialize();
```

```
    $this->loadComponent('Flash');
```

```
    // 新增以下行以檢查認證結果並保護網站
```

```
    $this->loadComponent('Authentication.Authentication');
```

```
}
```

新增登入功能

- 現在，每次請求時，**AuthenticationMiddleware** 都會檢查請求的會話以尋找已驗證的使用者。如果我們正在加載 `/users/login` 頁面，它也會檢查提交的表單數據（如果有）以提取憑證。預設情況下，憑證將從請求數據中的 `username` 和 `password` 欄位中提取。認證完成後，**Authentication Plugin** 會把認證結果放進 **Request** 的 `authentication` 屬性，因此 **Controller** 可以透過：

```
$this->request->getAttribute('authentication')
```

取得目前的認證狀態。

- 由於 **AuthenticationComponent** 在每次請求中都會檢查結果，因此所有頁面都會受到限制。如果未找到任何已驗證的使用者，它將把使用者重定向到 `/users/login` 頁面。請注意，此時網站尚無法正常運作，因為我們還沒有建立登入頁面。如果訪問您的網站，將會發生「無限重定向循環」，我們需要修復此問題。



如果您的應用程式同時透過 **SSL** 和 **non-SSL** 協議提供服務，則可能會出現會話遺失的問題，特別是在應用程式運行於非 **SSL** 協議的情況下。您需要透過設定 `config/app.php` 或 `config/app_local.php` 中的 `session.cookie_secure` 為 `false` 來啟用存取。（請參考 **CakePHP** 的 `session.cookie_secure` 預設值）

新增登入功能

在 UsersController 中新增以下程式碼：

```
// 檔案: src/Controller/UsersController.php
public function beforeFilter(\Cake\Event\EventInterface $event): void
{
    parent::beforeFilter($event);
    // 配置 login 動作無需認證，避免無限重定向問題
    $this->Authentication->allowUnauthenticated(['login']);
}

public function login()
{
    $result = $this->Authentication->getResult();
    // 如果使用者已登入，則將其重新導向至其他頁面。
    if ($result && $result->isValid()) {
        $target = $this->Authentication->getLoginRedirect() ?? [
            'controller' => 'Articles',
            'action' => 'index',
        ];
        return $this->redirect($target);
    }
    if ($this->request->is('post')) {
        $this->Flash->error(__('無效的帳號或密碼。'));
    }
}
```

UsersController.php

新增登入功能

在 /templates/Users/login.php 中新增以下內容：

```
<!-- 檔案: /templates/Users/Login.php -->
<div class="users form">
    <?= $this->Flash->render() ?>
    <h3>登入</h3>
    <?= $this->Form->create() ?>
    <fieldset>
        <legend><?= __('請輸入您的帳號與密碼') ?></legend>
        <?= $this->Form->control('email', ['required' => true]) ?>
        <?= $this->Form->control('password', ['required' => true]) ?>
    </fieldset>
    <?= $this->Form->submit(__('登入')) ?>
    <?= $this->Form->end() ?>

    <?= $this->Html->link("新增使用者", ['action' => 'add']) ?>
</div>
```

Login.php

新增登入功能

測試你的應用程式

1. 如果尚未建立使用雜湊密碼的使用者，請暫時註解掉與認證相關的程式碼，例如 `$this->loadComponent('Authentication.Authentication')`。
 2. 前往 `/users/add` 新增一個新使用者，輸入 `email` 和密碼。
 3. 確保新增後，取消註解所有與認證相關的程式碼。
- 試著訪問 `/articles/add` (尚未登入)。你應該會被重定向至登入頁，成功登入後，系統將自動將你重定向回 `/articles/add`。

新增登入功能

我們需要再添加一些細節來配置應用程式。我們希望所有的 `view` 和 `index` 頁面在未登入的情況下也能被訪問，為此我們將在 `AppController` 中添加以下特定的配置：

```
// 檔案: src/Controller/AppController.php
public function beforeFilter(\Cake\Event\EventInterface $event): void
{
    parent::beforeFilter($event);
    // 對於應用程式中的所有控制器，將 index 和 view 動作設為公開，跳過驗證檢查
    $this->Authentication->allowUnauthenticated(['index', 'view']);
}
```

AppController.php

新增登出功能

將登出動作添加到 UsersController 類別中：

```
// 在 src/Controller/UsersController.php 中  
public function logout()  
{  
    $this->Authentication->logout();  
    return $this->redirect(['controller' => 'Users', 'action' => 'login']);  
}
```

UserController.php

啟用註冊功能

如果你嘗試訪問 `/users/add` 而未登入，將會被重導到登入頁面。我們應該修復這個問題，因為我們希望允許人們註冊我們的應用程式。在 `UserController` 中修改以下程式碼：

```
// 在 src/Controller/UsersController.php 中 UserController.php  
$this->Authentication->allowUnauthenticated(['login', 'add']);
```

以上程式碼告訴 `AuthenticationComponent`，`UserController` 的 `add()` 不需要通過 `Authentication` 檢查。

授權

現在使用者已能登入我們的 CMS，我們希望應用授權規則，確保每位使用者只能編輯他們自己擁有的文章。我們將使用授權套件來實現這一功能。

安裝授權套件

使用 Composer 安裝授權套件：

```
composer require "cakephp/authorization:^3.0"
```

bash

在 src/Application.php 中的 bootstrap() 方法中加入以下語句來載入套件：

```
$this->addPlugin('Authorization');
```

Application.php

啟用授權套件

授權套件以中介軟體層的形式整合到您的應用程式中，並可選擇性地使用元件來簡化授權檢查的操作。首先，我們需要應用中介軟體。在 `src/Application.php` 中，將以下語句新增到類別的匯入部分：

```
use Authorization\AuthorizationService;  
use Authorization\AuthorizationServiceInterface;  
use Authorization\AuthorizationServiceProviderInterface;  
use Authorization\Middleware\AuthorizationMiddleware;  
use Authorization\Policy\OrmResolver;
```

Application.php

啟用授權套件

接著，在 Application 的已實現介面中，新增 AuthorizationServiceProviderInterface：

```
class Application extends BaseApplication
    implements AuthenticationServiceProviderInterface,
    AuthorizationServiceProviderInterface
{
```

Application.php

然後，在 middleware() 方法中加入以下內容：

```
// 在身份驗證之後添加授權
$middlewareQueue->add(new AuthorizationMiddleware($this));
```

Application.php

啟用授權套件

`AuthorizationMiddleware` 在處理請求時，將調用應用程式中的一個鉤點方法。此方法允許應用程式定義它希望使用的 `AuthorizationService`。在 `src/Application.php` 中新增以下方法：

Application.php

```
public function getAuthorizationService(ServerRequestInterface $request): AuthorizationServiceInterface
{
    $resolver = new OrmResolver();
    return new AuthorizationService($resolver);
}
```

`OrmResolver` 讓授權套件能找到 `ORM` 實體與查詢的策略類別。對於其他資源類型，也可以使用其他的解析器 (`Resolver`)。

啟用授權套件

接下來，我們需要將 `AuthorizationComponent` 加入到 `AppController` 中。在 `src/Controller/AppController.php` 的 `initialize()` 方法中，加入以下內容：

```
$this->loadComponent('Authorization.Authorization');
```

`AppController.php`

最後，我們需要將新增、登入和登出動作標記為不需要授權檢查。在 `src/Controller/UsersController.php` 中，為 `add`、`login` 和 `logout` 方法新增以下內容：

```
// 在 add、login 和 logout 方法中  
$this->Authorization->skipAuthorization();
```

`UserController.php`

`skipAuthorization()` 方法應該在任何控制器動作中調用，這些動作應該允許所有使用者（即使尚未登入）訪問。

建立我們的第一個授權策略

Authorization Plugin 使用 **Policy (策略) 類別** 定義授權規則。Policy 負責判斷「目前使用者是否可以對某個資源執行指定操作」。

我們的身份將是已登入的使用者，而資源則是我們的 **ORM** 實體和查詢。接下來，我們使用 **bake** 來生成一個基本的策略類別：

```
bin/cake bake policy --type entity Article
```

bash

建立我們的第一個授權策略

此指令將為我們的 `Article` 實體生成一個空的策略類別。生成的策略類別會位於 `src/Policy/ArticlePolicy.php`。接下來，將策略類別更新為以下內容：

```
<?php
namespace App\Policy;

use App\Model\Entity\Article;
use Authorization\IdentityInterface;

class ArticlePolicy
{
    public function canAdd(IdentityInterface $user, Article $article): bool
    {
        // 所有已登入的使用者都可以建立文章
        return true;
    }

    public function canEdit(IdentityInterface $user, Article $article): bool
    {
        // 已登入的使用者只能編輯自己的文章
        return $this->_isAuthor($user, $article);
    }
}
```

ArticlePolicy.php

建立我們的第一個授權策略

此指令將為我們的 `Article` 實體生成一個空的策略類別。生成的策略類別會位於 `src/Policy/ArticlePolicy.php`。接下來，將策略類別更新為以下內容：

`ArticlePolicy.php`

```
public function canDelete(IdentityInterface $user, Article $article): bool
{
    // 已登入的使用者只能刪除自己的文章
    return $this->_isAuthor($user, $article);
}

protected function _isAuthor(IdentityInterface $user, Article $article): bool
{
    return $article->user_id === $user->getIdentifier();
}
}
```

雖然我們在這裡定義了一些非常簡單的規則，但您可以根據應用程式的需求，在策略中使用更為複雜的邏輯。

在 ArticlesController 中檢查授權

啟用 AuthorizationComponent 後，每個需要處理的 Controller Action 都必須明確執行 `authorize()` 或 `skipAuthorization()`；若兩者都沒有呼叫，Authorization Plugin 會拋出例外，提醒我們漏掉授權檢查。

在 `src/Controller/ArticlesController.php` 中，將以下程式碼新增到 `add`、`edit` 和 `delete` 方法中：

```
public function add()
{
    $article = $this->Articles->newEmptyEntity();
    $this->Authorization->authorize($article);
    // 方法的其餘部分
}
```

ArticlesController.php

在 ArticlesController 中檢查授權

啟用 AuthorizationComponent 後，每個需要處理的 Controller Action 都必須明確執行 authorize() 或 skipAuthorization()；若兩者都沒有呼叫，Authorization Plugin 會拋出例外，提醒我們漏掉授權檢查。

在 src/Controller/ArticlesController.php 中，將以下程式碼新增到 add、edit 和 delete 方法中：

```
public function edit($slug)
{
    $article = $this->Articles
        ->findBySlug($slug)
        ->contain('Tags') // 載入相關的 Tags
        ->firstOrFail();
    $this->Authorization->authorize($article);
    // 方法的其餘部分
}
```

ArticlesController.php

在 ArticlesController 中檢查授權

啟用 AuthorizationComponent 後，每個需要處理的 Controller Action 都必須明確執行 authorize() 或 skipAuthorization()；若兩者都沒有呼叫，Authorization Plugin 會拋出例外，提醒我們漏掉授權檢查。

在 src/Controller/ArticlesController.php 中，將以下程式碼新增到 add、edit 和 delete 方法中：

```
public function delete($slug)
{
    $this->request->allowMethod(['post', 'delete']);

    $article = $this->Articles->findBySlug($slug)->firstOrFail();
    $this->Authorization->authorize($article);
    // 方法的其餘部分
}
```

ArticlesController.php

在 ArticlesController 中檢查授權

`AuthorizationComponent::authorize()` 方法將使用當前的控制器動作名稱來生成需要呼叫的策略方法。如果您希望呼叫不同的策略方法，可以使用操作名稱來呼叫 `authorize`：

```
$this->Authorization->authorize($article, 'update');
```

在 ArticlesController 中檢查授權

最後，將以下程式碼新增到 ArticlesController 中的 tags、view 和 index 方法：

```
// view、index 和 tags 方法是公開方法  
// 不需要進行授權檢查。  
$this->Authorization->skipAuthorization();
```

ArticlesController.php

修正新增和編輯動作

雖然我們已經透過 `Policy` 限制使用者只能進入自己文章的 `edit()`，但如果 `user_id` 仍允許 `Mass Assignment`，使用者仍可能在提交資料時嘗試修改文章擁有者。我們接下來將解決這些問題。首先是新增動作。

在創建文章時，我們希望將 `user_id` 設定為當前登錄的用戶。將 `add` 動作替換為以下程式碼：

// 在 `src/Controller/ArticlesController.php` 中

`ArticlesController.php`

```
public function add()
{
    // ...

    if ($this->request->is('post')) {
        $article = $this->Articles->patchEntity($article, $this->request->getData());

        // 更改：將 user_id 設為當前用戶。
        $article->user_id = $this->request->getAttribute('identity')->getIdentifier();

        // ...
    }
}
```

修正新增和編輯動作

接下來，我們將更新 `edit` 動作。將 `edit` 方法替換為以下程式碼：

// 在 `src/Controller/ArticlesController.php` 中

`ArticlesController.php`

```
public function edit($slug)
{
    // ...

    if ($this->request->is(['post', 'put'])) {
        $this->Articles->patchEntity($article, $this->request->getData(), [
            // 新增：禁用修改 user_id。
            'accessibleFields' => ['user_id' => false]
        ]);
    }

    // ...
}
```

在這裡，我們通過 `patchEntity()` 的選項來修改哪些屬性可以進行批量賦值。欲了解更多資訊，請參見[更改可訪問字段](#)。請記得從 `templates/Articles/edit.php` 中刪除 `user_id` 控制項，因為我們不再需要它。

Exercise：完善 Users 與 Tags 的授權控制

- 目前我們已經為 `Articles` 建立基本的 `Authorization Policy`，限制使用者只能編輯或刪除自己擁有的文章。
- 請參考 `ArticlePolicy` 與 `ArticlesController` 的做法，將相同的授權概念延伸到 **Users** 與 **Tags**，完成 CMS 的權限控制。

Exercise：完善 Users 與 Tags 的授權控制

作業目標

請完成以下需求：

1. 為 **User** 與 **Tag** 建立對應的 **Policy** 類別。
2. 在 **UserController** 與 **TagsController** 中加入適當的授權檢查。
3. 需要授權的 **Action** 必須使用：

```
$this->Authorization->authorize(...);
```

4. 不需要授權的公開 **Action**，必須明確使用：

```
$this->Authorization->skipAuthorization();
```

5. 使用者只能編輯或刪除**自己的 User** 資料。
6. 一般使用者可以瀏覽 **Tags**，但修改 **Tags** 的操作必須加入適當的權限限制。
7. 不應僅透過隱藏表單欄位或前端介面限制操作，必須在後端透過 **Policy** 進行授權判斷。
8. 完成後，請測試未登入、已登入但無權限、以及具備權限時的執行結果。

Exercise：完善 Users 與 Tags 的授權控制

建議實作項目

可參考 Day 5 中 `ArticlePolicy` 的結構：

```
ArticlePolicy
├── canAdd()
├── canEdit()
├── canDelete()
└── isAuthor()
```

自行思考 `UserPolicy` 與 `TagPolicy` 應該包含哪些規則，例如：

`UserPolicy`

- 誰可以新增使用者？
- 誰可以查看使用者？
- 使用者是否只能修改自己的帳號？
- 使用者是否只能刪除自己的帳號？

`TagPolicy`

- 誰可以新增標籤？
- 誰可以編輯標籤？
- 誰可以刪除標籤？
- 哪些標籤頁面可以公開瀏覽？

Exercise：完善 Users 與 Tags 的授權控制

完成條件

完成後，你的 CMS 應該至少能區分：

Authentication

→ 使用者是否已登入？

Authorization

→ 已登入的使用者是否有權限執行這個操作？

並讓：

Articles

Users

Tags

都具備明確的授權規則。

Exercise：完善 Users 與 Tags 的授權控制

挑戰題

嘗試避免在 Controller 中直接撰寫大量：

```
if ($user->id === ...)
```

的權限判斷。

將授權邏輯集中在：

```
src/Policy/
```

中，讓 Controller 專注於處理 Request 與 Application Flow。

這樣之後即使權限規則改變，也能直接修改 Policy，而不需要到每一個 Controller Action 中逐一調整。

結束語

我們已經建立了一個簡單的 CMS 應用程式，允許用戶登入、發布文章、為文章加入標籤、依標籤瀏覽文章，並加入基本的文章授權與擁有者存取控制。我們還利用了 FormHelper 和 ORM 的功能，為使用者體驗（UX）添加了一些最佳化。

感謝您花時間探索 CakePHP。接下來，您可以深入了解[資料庫存取與 ORM](#)，或者繼續閱讀《[使用 CakePHP](#)》的相關內容。