RBAC Authorization on an yii2-framework based system

2020/03/13 (Fri)

By Umbalaconmeogia

Table of Contents

Table of Contents

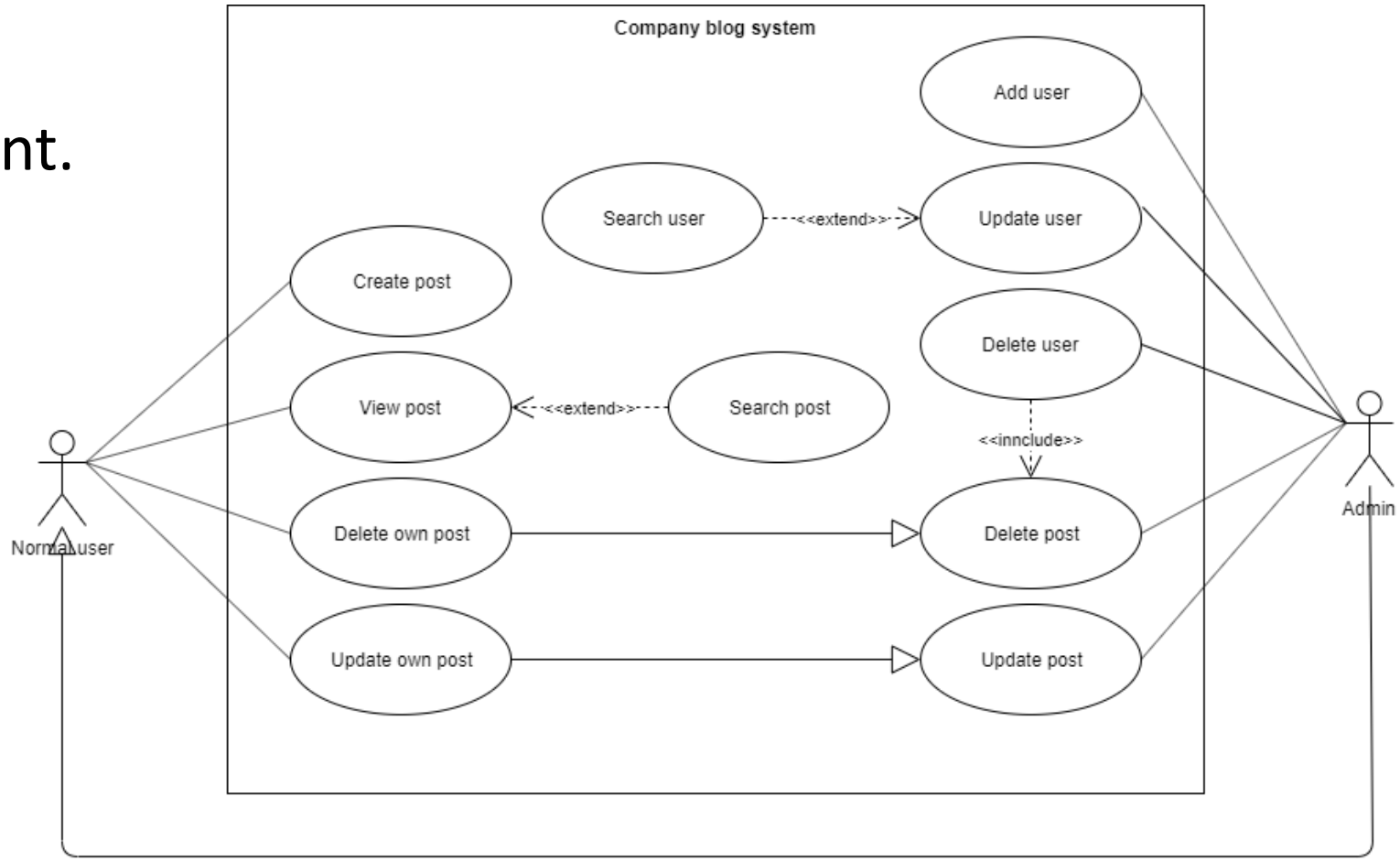
Overview

- Purpose of this document:
  Introduce about RBAC on yii2 framework, with some usage example.
- Prerequisite:
  - Audience should have basic experience of using yii2 framework.
  - Read about Authorization on yii2
- References:
  - Sample code on github.

Table of Contents

Sample project: Company blog system

- A simple system to post and view content.



Database design

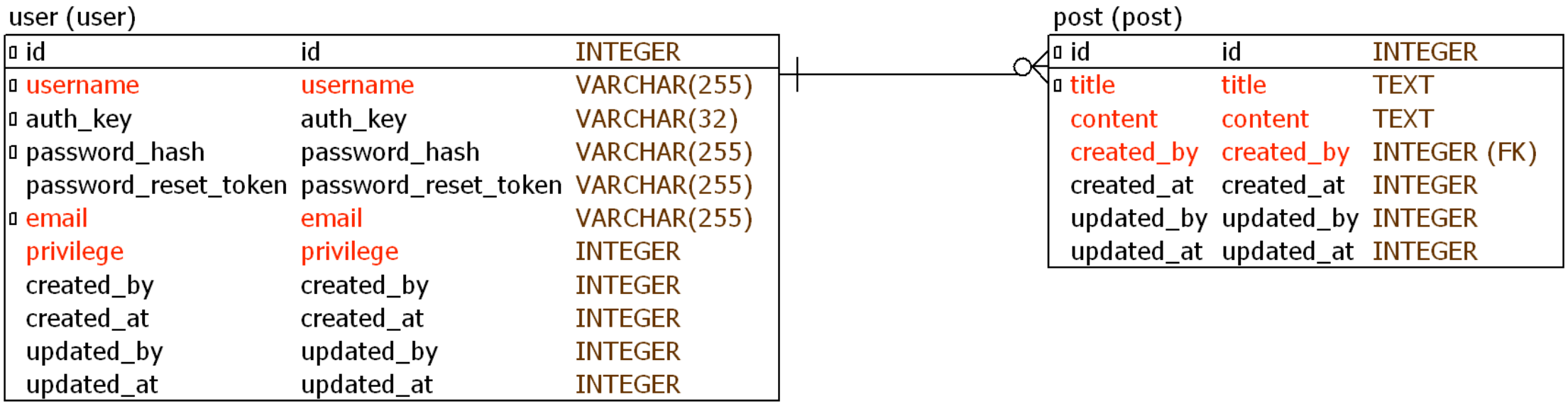


Table of Contents

Access Control Filter

- Decide if current web user can access to a controller's action.
- Filter defined in controller's behaviors()
- Reference:
  - Access Control Filter
  - Request handling overview

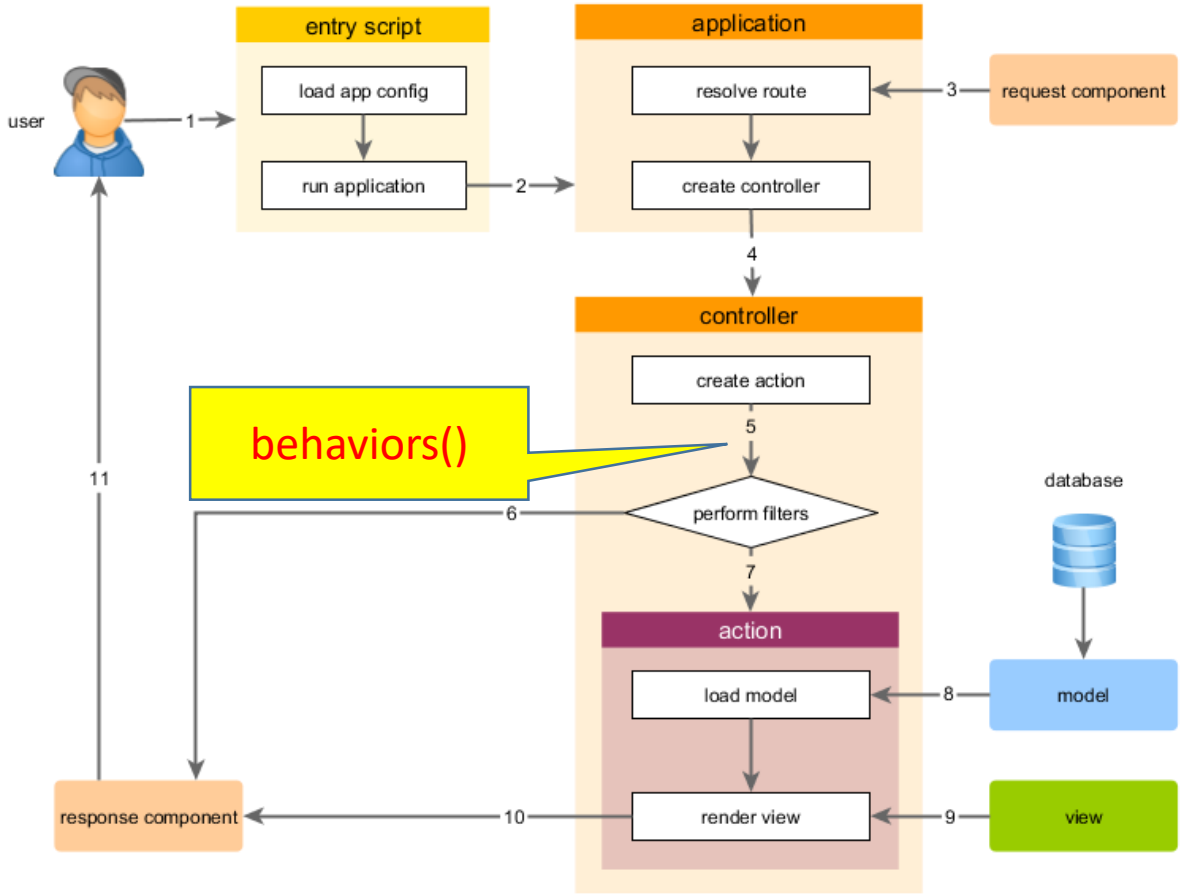


Table of Contents

Why we want to use a framework like RBAC

Default function of ACF is for simple case only.
Some easy thinking code like this ☺

```
if ($post->created_by == Yii::$app->user->id || Yii::$app->user->privilege = User::PRIVILEGE_ADMIN) {
    // Do update post
}
```

And we need to write code like this many times in the code to check privilege

By using a good framework, it become short and clear.

```
if (Yii::$app->user->can('updatePost')) {
    // Do update post
}
```

1. **We concentrate into writing business code (do update post), leave privilege checking to the framework.**
2. **By using RBAC combining with ACF, we even move privilege checking out of the business logic code.**
3. **It's easier to maintain (adding more privileges, role…).**

RBAC

- RBAC is a method to build access control.
- It builds a way to check if current user have permission to do something.
  - RBAC can be used as filter in ACF.
  - RBAC can be used in any place of code.
- 2 steps to use RBAC:
  1. Build RBAC data (define permission/role/rule and assign permission).
  2. Use RBAC in code.
- Reference: Role Based Access Control

RBAC used in ACF

Simple roles defined in ACF

```
public function behaviors()
{
    return [
        'access' => [
            'class' => AccessControl::className(),
            'only' => ['login', 'logout', 'signup'],
            'rules' => [
                [
                    'allow' => true,
                    'actions' => ['login', 'signup'],
                    'roles' => ['?'],
                ],
                [
                    'allow' => true,
                    'actions' => ['logout'],
                    'roles' => ['@'],
                ],
            ],
        ],
    ];
}
```

ACF using roles defined by RBAC

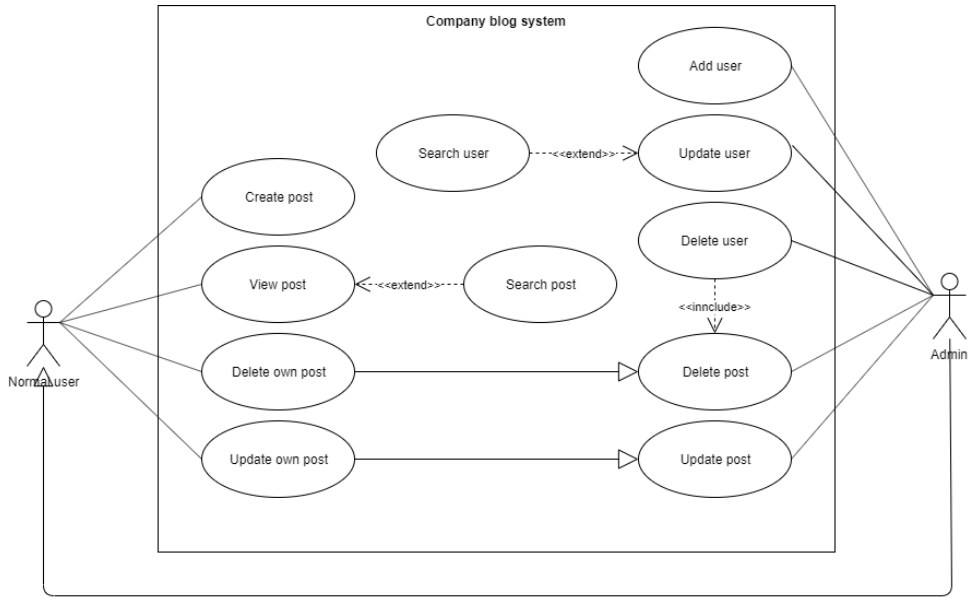
```
public function behaviors()
{
    return [
        'access' => [
            'class' => AccessControl::className(),
            'rules' => [
                [
                    'allow' => true,
                    'actions' => ['index'],
                    'roles' => ['managePost'],
                ],
                [
                    'allow' => true,
                    'actions' => ['view'],
                    'roles' => ['viewPost'],
                ],
                [
                    'allow' => true,
                    'actions' => ['create'],
                    'roles' => ['createPost'],
                ],
                [
                    'allow' => true,
                    'actions' => ['update'],
                    'roles' => ['updatePost'],
                ],
                [
                    'allow' => true,
                    'actions' => ['delete'],
                    'roles' => ['deletePost'],
                ],
            ],
        ],
    ];
}
```

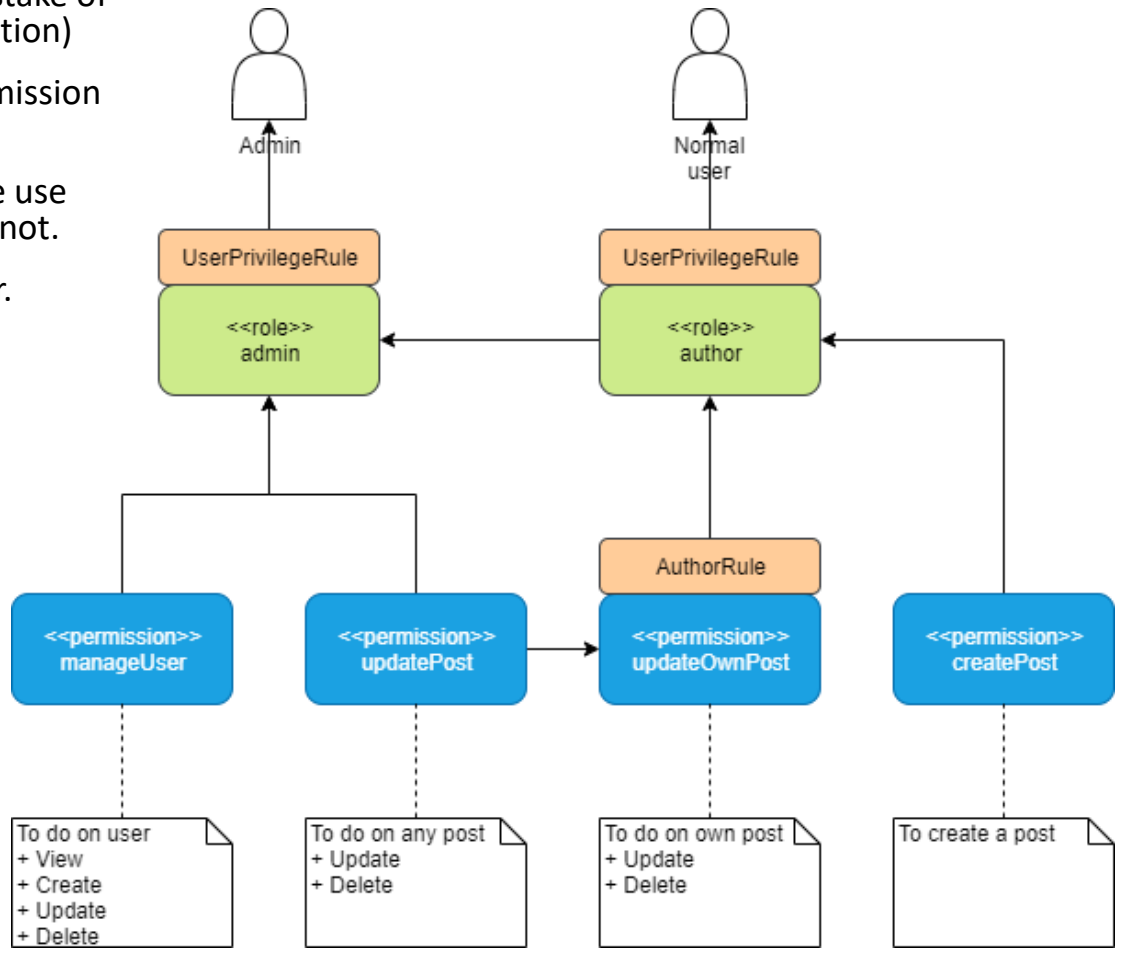
RBAC used in code

```
if (¥Yii::$app->user->can('createPost')) {
    // create post
}

if (¥Yii::$app->user->can('updatePost', ['post' => $post])) {
    // update post
}
```

RBAC data design

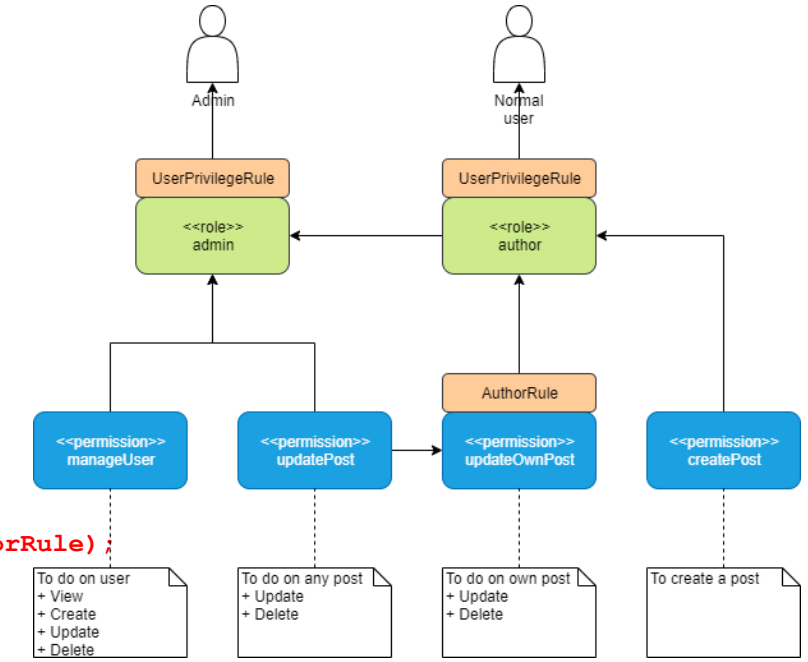
- RBAC data doesn't relate to system function (it may be common mistake of access control design to create access permission same as user function)
- Any logged in user can view post, so it is unnecessary to define permission for viewing post (we have AFC definition for viewing post).
- We don't assign permission/role to specified user in this system. We use rule (based on User#privilege) to determine if a user is an admin or not.
- "admin" and "author" is default roles, which is applied to every user.





RBAC permission and role

```
public function actionInit()
{
    $auth = Yii::$app->authManager;
    $auth->removeAll();

    // add the rule "author"
    $authorRule = new AuthorRule();
    $auth->add($authorRule);

    // add "createPost" permission
    $createPost = $this->createPermission('createPost', 'Create a post');

    // add "updatePost" permission
    $updatePost = $this->createPermission('updatePost', 'Update a post');

    // add "updateOwnPost" permission
    $updateOwnPost = $this->createPermission('updateOwnPost', 'Update own post', $authorRule);
    // "updateOwnPost" will be used from "updatePost"
    $auth->addChild($updateOwnPost, $updatePost);

    // add "author" role and give this role the "createPost", "updateOwnPost" permissions
    $author = $this->createRole('author', [$createPost, $updateOwnPost]);

    // add the rule "userPrivilege"
    $userPrivilegeRule = new UserPrivilegeRule();
    $auth->add($userPrivilegeRule);

    // add "manageUser" permission
    $manageUser = $this->createPermission('manageUser', 'Manage user');

    // add "admin" role and give this role the "manageUser", "updatePost" permission
    // as well as the permissions of the "author" role
    $admin = $this->createRole('admin', [$manageUser, $updatePost, $author], $userPrivilegeRule);
}
```



Source Code

RBAC rule

- A rule is just for checking if current login user (specified by id) is matching with specified role/permission (difficult to understand).

Check for Author

```
class AuthorRule extends Rule
{
    public $name = 'author';

    /**
     * @param string|int $user the user ID.
     * @param Item $item the role or permission
     * @param array $params parameters passed to
     *              ManagerInterface::checkAccess().
     * @return bool
     */
    public function execute($user, $item, $params)
    {
        return isset($params['post']) ?
            $params['post']->created_by == $user : false;
    }
}
```

Source Code

Check for admin privilege

```
class UserPrivilegeRule extends Rule
{
    public $name = 'userPrivilege';

    public function execute($user, $item, $params)
    {
        if (!Yii::$app->user->isGuest) {
            $privilege = Yii::$app->user->identity->privilege;
            if ($item->name === 'admin') {
                return $privilege == User::PRIVILEGE_ADMIN;
            } elseif ($item->name === 'author') {
                return $privilege == User::PRIVILEGE_ADMIN
                    || $privilege == User::PRIVILEGE_NORMAL;
            }
        }
        return false;
    }
}
```

Source Code

Table of Contents

Demo system

System setup

```
# Install depencencies.
composer install

# Create database
./yii migrate

# Create sample data for user and post
./yii initial-data
```

This system has two users

```
Admin
Account: user1, Password: password1

Normal user
Account: user2, Password: password2
```

Access control on actions (using ACF)

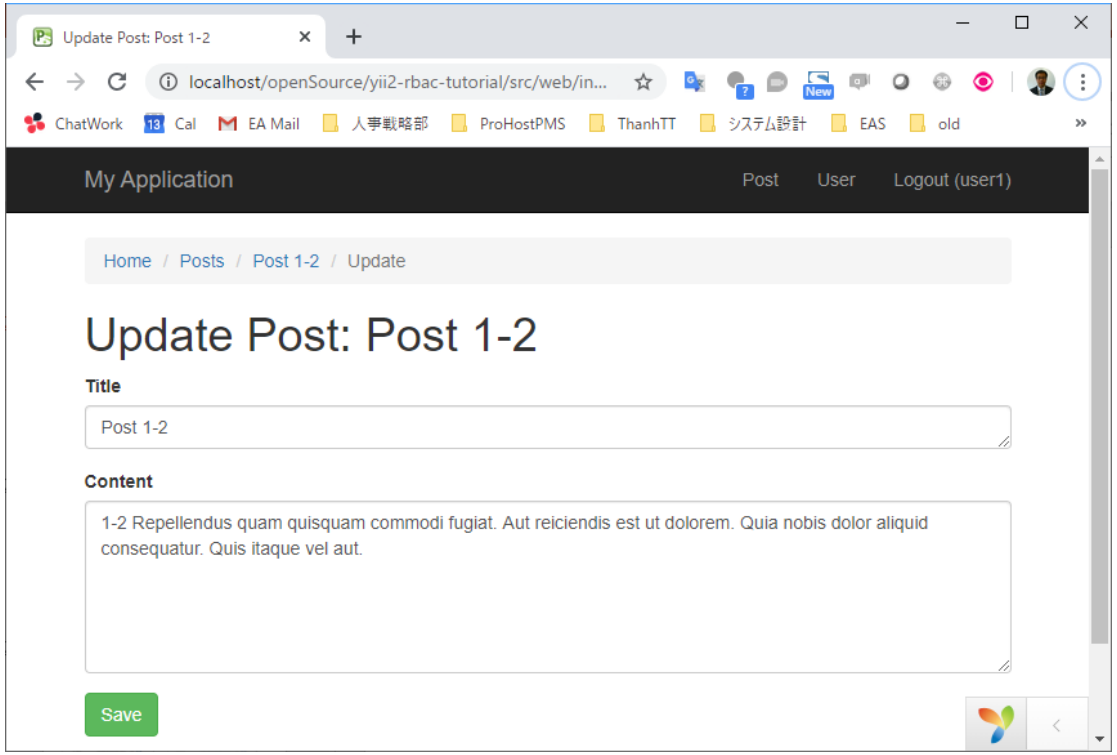
```
class PostController extends Controller
{
    public function behaviors()
    {
        return [
            'access' => [
                'class' => AccessControl::className(),
                'rules' => [
                    [
                        'allow' => true,
                        'actions' => ['index', 'view'],
                        'roles' => ['@'],
                    ],
                    [
                        // In fact, we don't need to define this rule here, but can defined it in roles = @ above.
                        // I put it here just for an example.
                        'allow' => 'true',
                        'actions' => ['create'],
                        'roles' => ['createPost'],
                    ],
                    [
                        'allow' => true,
                        'actions' => ['update', 'delete'],
                        'roles' => ['updatePost'],
                        'roleParams' => function() {
                            return ['post' => Post::findOne(['id' => Yii::$app->request->get('id')])];
                        },
                    ],
                ],
            ],
        ];
    }
}
```

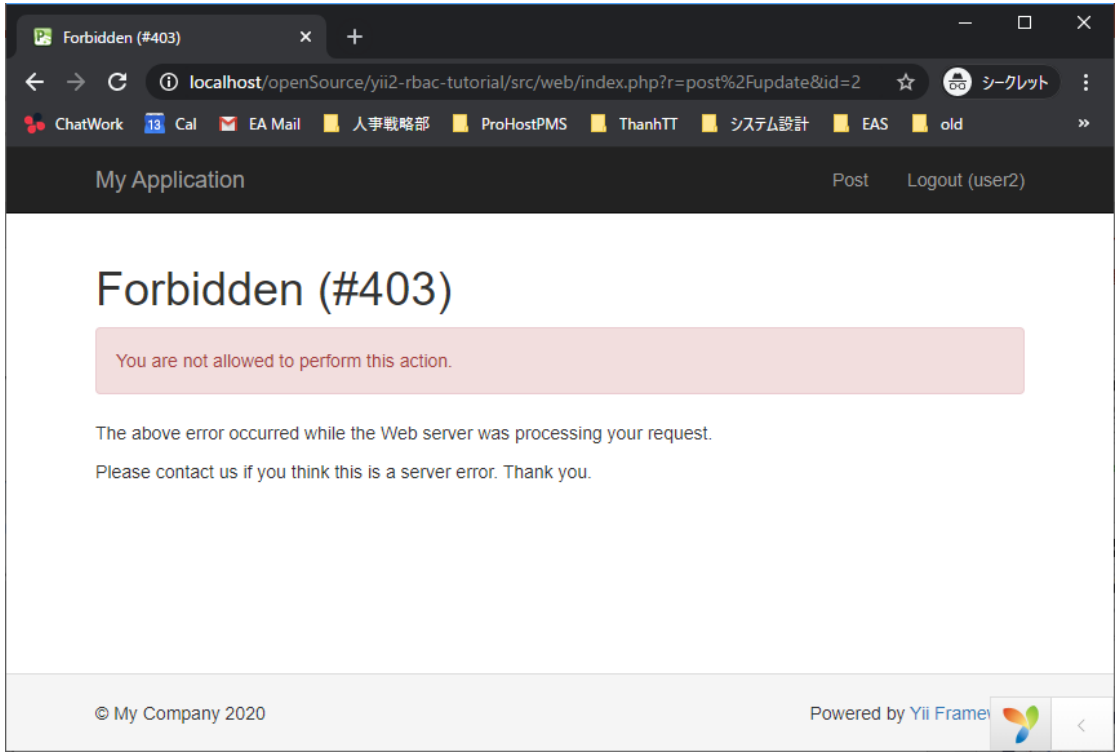
‘roles’ is ACF keyword, not relate to RBAC role concept

This will be checked for both admin and author roles

Source Code

Demo of RBAC used with ACF





user1 can edit/delete any user's post.

user2 cannot edit/delete other user's post.

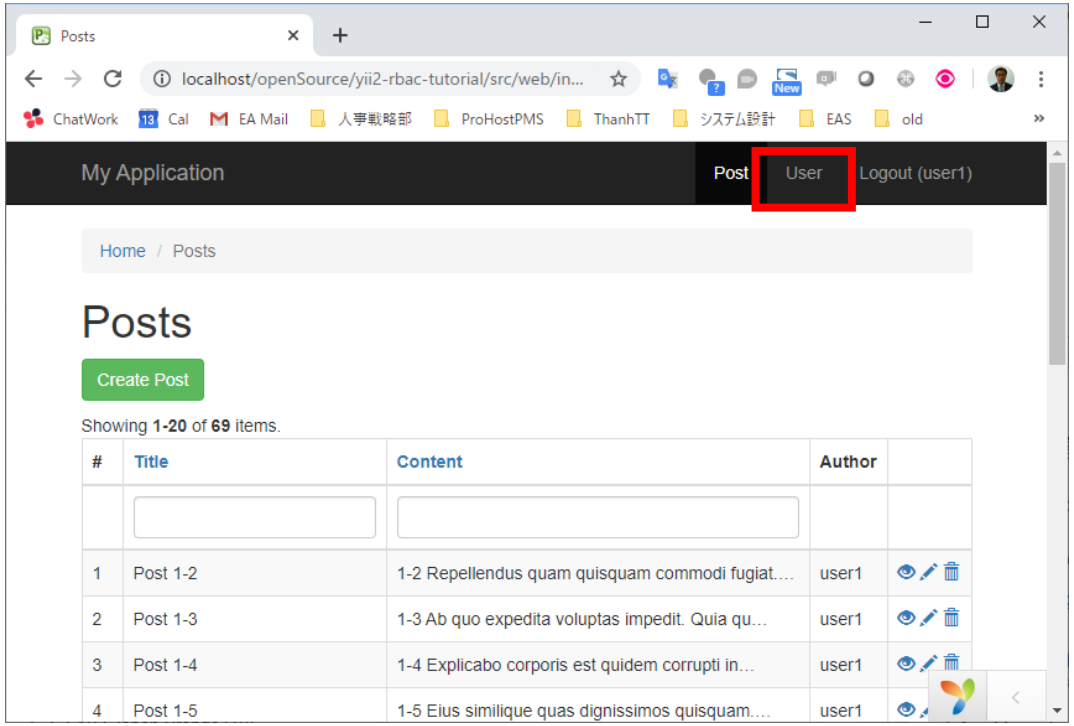
Access control applied on UI (1)

Source Code

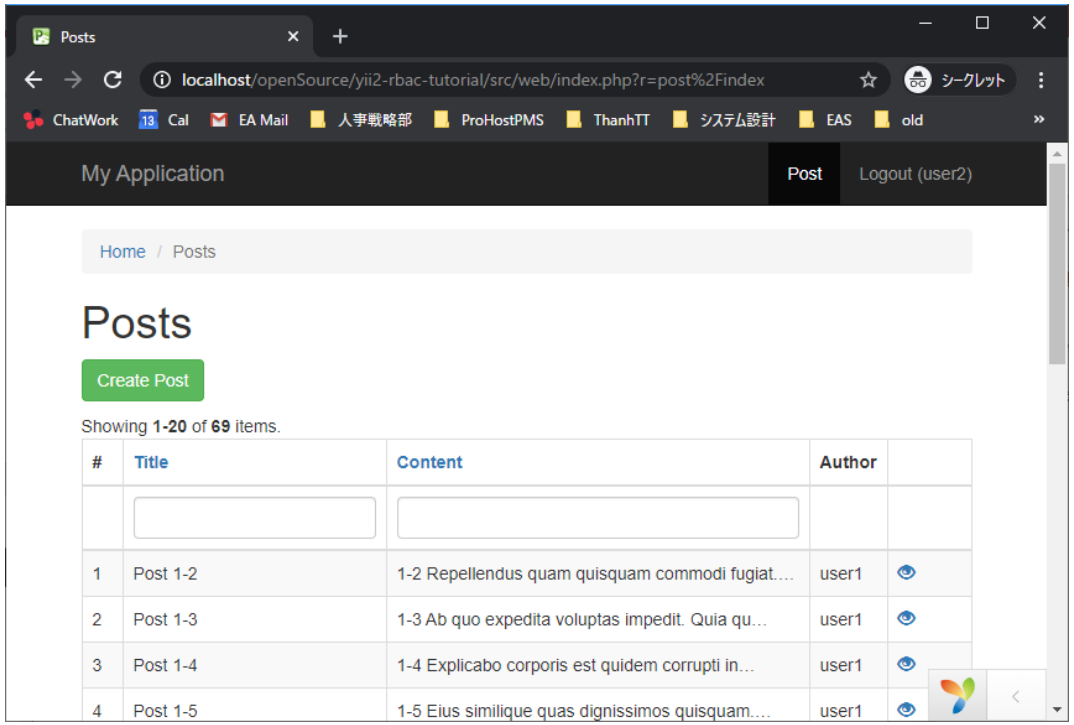
For menu bar

```
$items = [];
if (Yii::$app->user->isGuest) {
    $items[] = ['label' => 'Login', 'url' => ['/site/login']];
} else {
    $items[] = ['label' => 'Post', 'url' => ['/post/index']];
    if (Yii::$app->user->can('manageUser')) {
        $items[] = ['label' => 'User', 'url' => ['/user/index']];
    }
    $items[] = $logoutMenuItem;
}
```

Demo of menu



user1 can access to user management from menu



user2 can’t access to user management from menu

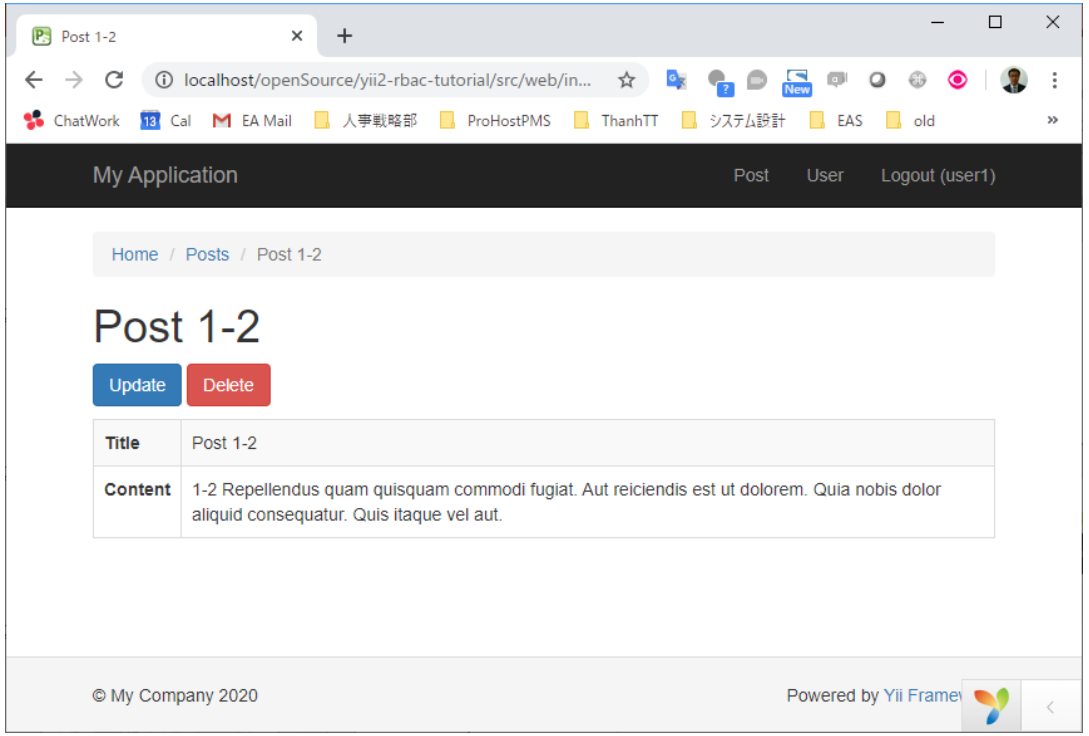
Access control applied on UI (2)

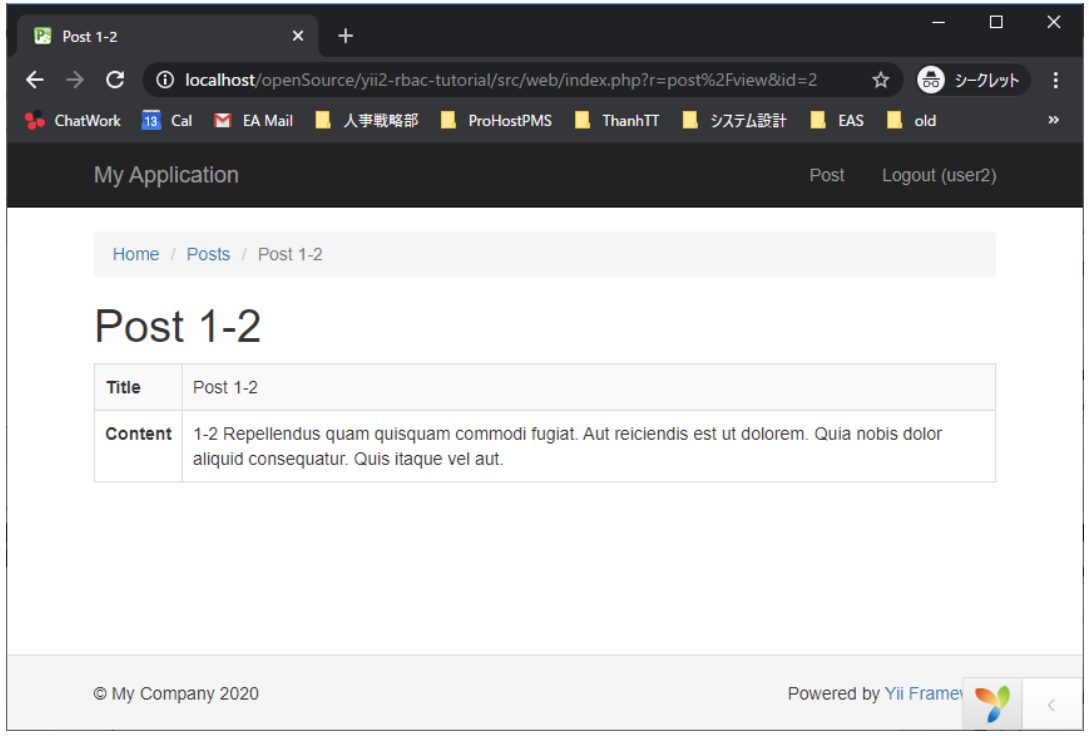
To control displaying of buttons on Post view page

Source Code

```
<?php if (Yii::$app->user->can('updatePost')) { ?>
    <?= Html::a('Update', ['update', 'id' => $model->id], ['class' => 'btn btn-primary']) ?>
    <?= Html::a('Delete', ['delete', 'id' => $model->id], [
        'class' => 'btn btn-danger',
        'data' => [
            'confirm' => 'Are you sure you want to delete this item?',
            'method' => 'post',
        ],
    ]) ?>
<?php } ?>
```

Demo of buttons





user1 has button to edit/delete any user's post.

user2 can not edit/delete other user's post.

Behavior based on access permission

- **`Use Yii::$app->user->can() on business logic code`**

The End